Τα τυπικά του μαθήματος

ΠΕΡΙΕΧΟΜΕΝΟ ΜΑΘΗΜΑΤΟΣ

Εισαγωγή στην ηλεκτρονική, ψηφιακές διατάξεις, κυκλώματα και συστήματα. Ψηφιακά σήματα. Μέθοδοι κωδικοποίησης ψηφιακών δεδομένων. Λογικές πύλες. Άλγεβρα Boole, λογικές συναρτήσεις, κανονική παράσταση συναρτήσεως. Μέθοδοι απλοποίησης συναρτήσεων, χάρτες Karnaugh, αλγεβρικές μέθοδοι. Συνδυαστικά κυκλώματα. Μετρητές, συγκριτές, κωδικοποιητές, αθροιστές κλπ. Ακολουθιακά κυκλώματα, σύγχρονα, ασύγχρονα κυκλώματα. Μανδαλωτές, flip-flop, μνήμες ROM & RAM. Εισαγωγή στις γλώσσες περιγραφής υλικού (Verilog, VHDL).

Προτεινόμενη βιβλιογραφία

•**Mano Morris, Ciletti Michael, "Ψηφιακή Σχεδίαση", 4η έκδοση, (Μετάφραση), Εκδόσεις Παπασωτηρίου, 2010.**

•**J.F.Wakerly, "Ψηφιακή Σχεδίαση: Αρχές & Πρακτικές", 3η έκδοση,** (Μετάφραση) **Εκδόσεις Κλειδάριθμος ΕΠΕ, 2002**

Ερωτήματα:

Α) Τι είναι ένας Υπολογιστής;

Β) Τι είναι ένας Ηλεκτρονικός Υπολογιστής;

Γ) Τι είναι η Ηλεκτρονική;;;

<u>ΗΛΕΚΤΡΟΝΙΚΗ</u>

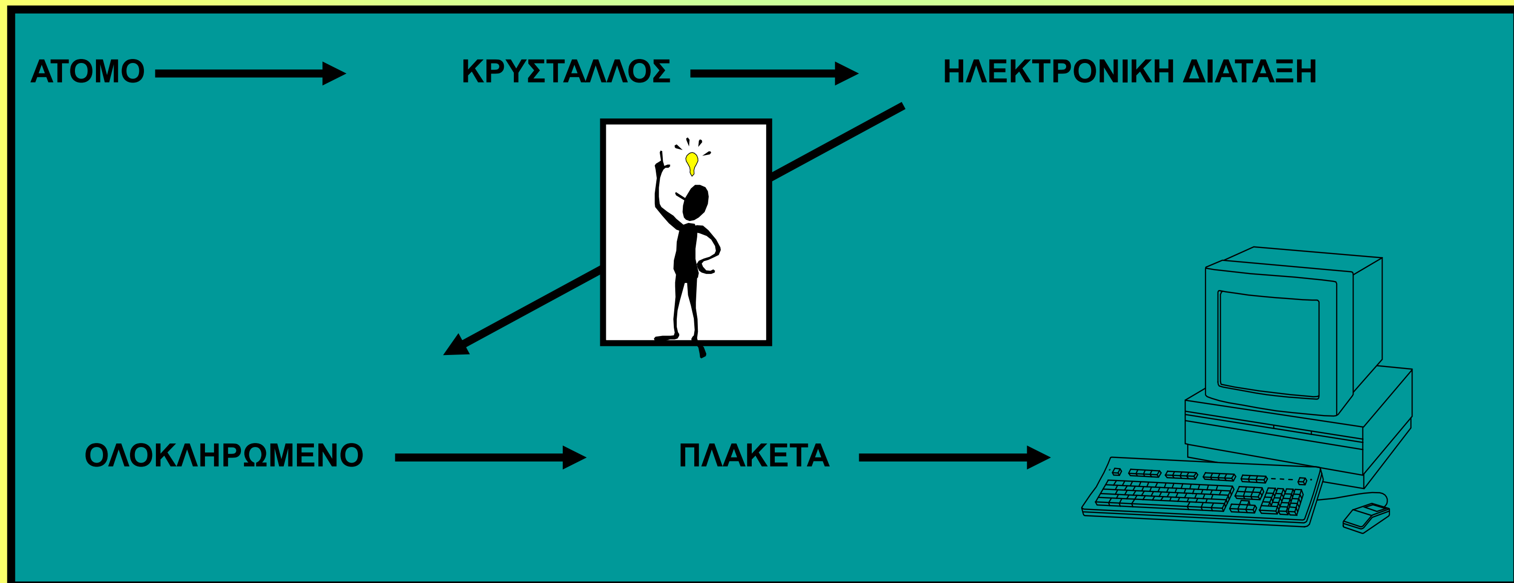
Το σύνολο των τεχνικών που χρησιμοποιούν

τις **μεταβολές** **<u>ηλεκτρικών μεγεθών</u>**

(Ηλεκτρομαγνητικών πεδίων, πληθυσμών φωτονίων και ηλεκτρικών φορέων)

για να ***<u>συλλάβουν, να διαβιβάσουν και να εκμεταλλευτούν</u>***

μια **<u>πληροφορία.</u>**



Η *<u>πληροφορία</u>* μπορεί να είναι άμεσα αντιληπτή από εμάς, όπως μια φωνή ή μια εικόνα μπορεί όμως και να την αντιλαμβανόμαστε με την βοήθεια οργάνων που εμείς έχουμε επινοήσει και κατασκευάσει όπως το ραδιόφωνο, η τηλεόραση ή και ακόμα πιο εξελιγμένες συσκευές όπως το radar ή οι δορυφόροι.

Η πληροφορία, (το μήνυμα) μπορεί να είναι δημιούργημα του ανθρώπου όπως στις διάφορες τηλεπικοινωνίες, μπορεί όμως να είναι ένα μήνυμα από κάποιο φυσικό φαινόμενο, μικροσκοπικό ή μακροσκοπικό, όπως για παράδειγμα το ράδιο-αστρονομικό μήνυμα των εκπομπών από τον Ήλιο ή μηνύματα από μικροσκοπικά φαινόμενα όπως η βιολογική πληροφορία που απασχολεί τα ιατρικά ηλεκτρονικά.

ΙΣΤΟΡΙΑ ΤΗΣ ΗΛΕΚΤΡΟΝΙΚΗΣ

✓Το 1875 ο Bell σκέφτηκε πρώτος να χρησιμοποιήσει την αγωγιμότητα για μετάδοση με καλώδιο από μακριά των διακυμάνσεων του ρεύματος ενός μικροφώνου. Λίγο αργότερα ο Hertz επιδίωξε το ίδιο πράγμα χωρίς καλώδιο, με τη βοήθεια ηλεκτρομαγνητικών κυμάτων, αλλά αυτό το πέτυχε πρώτος ο Μαρκόνι.

✓Η γέννηση της Ηλεκτρονικής τοποθετείται χρονικά γύρω στα 1902, όταν ο Flemming επινόησε την πρώτη δίοδο ηλεκτρονική λυχνία, ή το 1904, όταν ο Lee de Forest δημιούργησε την πρώτη τρίοδο. Κατά την διάρκεια τού Α' Παγκόσμιου Πολέμου, σημαντική ήταν η εξέλιξη της ασύρματης επικοινωνίας. Ακολούθησε η Ραδιοτηλεφωνία και η Ραδιοφωνία, το 1926. Η διαδρομή αυτή συμπληρώθηκε περί το 1935 με την τηλεόραση.

✓1930-1940: Ραντάρ, Πολυπλεξία στις επικοινωνίες (Multiplexing), Ραδιοπλοήγηση και αναπτύχθηκε πιο πέρα η μελέτη των συνθηκών διάδοσης των Ηλεκτρομαγνητικών κυμάτων κυρίως στα βραχέα και υπερβραχέα. Μαθηματικές μέθοδοι που περιγράφουν τη διαμόρφωση και γενικότερα την κωδικοποίηση. Διαμορφώσεις Πλάτους, Συχνότητας, Φάσης. (PAM, PPM, PDM) και η TDM. Γεννιέται και η Παλμοκωδική διαμόρφωση (PCM) . Εργασίες του Shannon για τη χωρητικότητα ενός καναλιού - Άριστοι κώδικες,

✓**Η μεγάλη τομή στην εξέλιξη της τεχνολογίας των ηλεκτρονικών συσκευών έγινε το 1948 με την κατασκευή του πρώτου διπολικού τρανζίστορ επαφής (BJT) από τον Bardeen και τους συνεργάτες του Schokley και Brattain στο Πανεπιστήμιο του Illinois.**

✓**Γύρω στα 1965 με 1966 έγινε η δεύτερη μεγάλη καινοτομία στο χώρο της Ηλεκτρονικής και των εφαρμογών της με την εμφάνιση των πρώτων ολοκληρωμένων κυκλωμάτων (Integrated Circuits – IC) από τον J.Kilby.**

- 1970- Σήμερα. Μικροσμίκρυνση (minaturisation) χάρη στα MOS (FET Metal Oxide Semiconductor).
- MSI (Medium Size Integrated: ολοκλήρωση σε μεσαία κλίμακα, δεκαετία του 60),
- LSI (δεκαετία του 70) (Large scale Integrated: ολοκλήρωση σε μεγάλη κλίμακα),
- VLSI την δεκαετία του ογδόντα (Very Large Scale Integration- Ολοκλήρωση Πολύ Μεγάλης κλίμακας)
- και την τελευταία δεκαετία τα ULSI (Ultra-Large Scale Integration- Ολοκλήρωση εξαιρετικά μεγάλης κλίμακας). Σ' αυτά οφείλονται οι υπολογιστές τσέπης και οι μικροϋπολογιστές.

✓Σήμερα η σχεδίαση και παραγωγή ηλεκτρονικών διατάξεων βρίσκεται στην περιοχή των ολοκληρωμένων κυκλωμάτων και μετατοπίζεται διαρκώς προς συνθετότερα συστήματα, που συγκεντρώνουν όλο και περισσότερες ηλεκτρονικές λειτουργίες σε ένα δομικό σύνολο (chip) δημιουργώντας τα Συστήματα σε Chip (System on a Chip).

ΗΛΕΚΤΡΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ

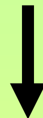
•**Παλαιότερα υπήρχαν οι ηλεκτρονικες δίοδοι, τρίοδοι κλπ λυχνίες κενού. Σήμερα βρίσκονται είτε στα μουσεία είτε σε ορισμένες συσκευές ισχύος.**

•Αντικαταστάθηκαν από τις **Διατάξεις Στερεάς Κατάστασης** (Solid State Devices). Η λειτουργία τους βασίζεται στις ιδιότητες των **ΗΜΙΑΓΩΓΩΝ**.

•Κατασκευάζονται **ΔΙΑΚΡΙΤΕΣ ΔΙΑΤΑΞΕΙΣ ή** και **ΟΛΟΚΛΗΡΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ (MICROCHIPS).**

+

ΠΑΘΗΤΙΚΑ ΣΤΟΙΧΕΙΑ
(R, L, C)



ΗΛΕΚΤΡΟΝΙΚΑ ΚΥΚΛΩΜΑΤΑ
(ΔΙΑΚΡΙΤΑ ή ΟΛΟΚΛΗΡΩΜΕΝΑ)

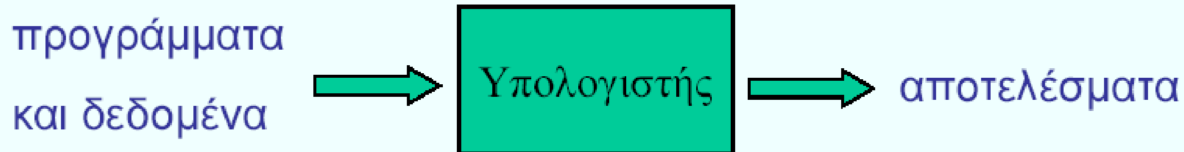
ΒΑΣΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΣΤΕΡΕΑΣ ΚΑΤΑΣΤΑΣΗΣ:

1. ΔΙΟΔΟΙ
2. ΔΙΠΟΛΙΚΑ ΤΡΑΝΖΙΣΤΟΡ ΕΠΑΦΗΣ
3. ΤΡΑΝΖΙΣΤΟΡ ΕΠΙΔΡΑΣΗΣ ΠΕΔΙΟΥ(FET).
4. ΟΠΤΟΗΛΕΚΤΡΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ (LED, LASERS Κ.Α.).

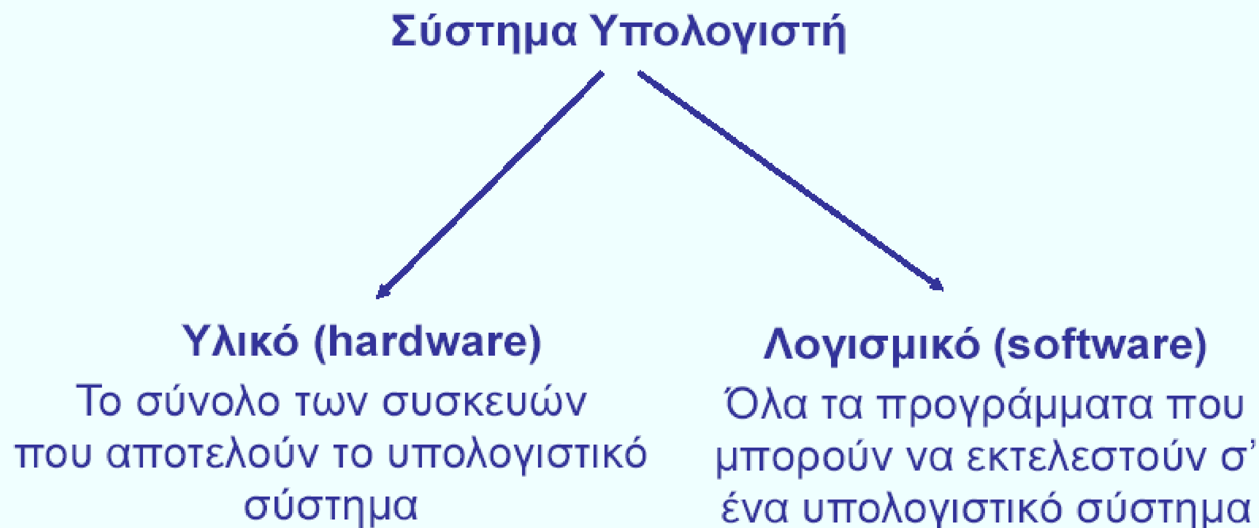
ΕΦΑΡΜΟΓΕΣ:

Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ, ΗΛΕΚΤΡΟΝΙΚΑ ΙΣΧΥΟΣ, ΔΙΑΣΤΗΜΙΚΕΣ ΕΦΑΡΜΟΓΕΣ, ΜΕΤΑΦΟΡΕΣ, ΣΥΣΚΕΥΕΣ ΚΑΘΗΜΕΡΙΝΗΣ ΧΡΗΣΗΣ Κ.Α.

Υπολογιστής: Σύστημα επεξεργασίας πληροφοριών



Σύστημα Υπολογιστή



K.M.E

Μονάδες Εισ/Εξ

Σύστημα διασύνδεσης

Σύστημα Μνήμης

Η δομή ενός προσωπικού υπολογιστή

Μικροεπεξεργαστής

ΚΜΕ

ΜΑΜ

Κρυφή μνήμη

Μονάδα διασύνδεσης αρτηρίας

Κύρια μνήμη

Μνήμη μαγνητικού δίσκου

Οθόνη

Πληκτρολόγιο

Δίκτυο επικοινωνίας

Ελεγκτής μονάδας δίσκου

Ελεγκτής οθόνης

Ελεγκτής πληκτρολογίου

Ελεγκτής δικτύου

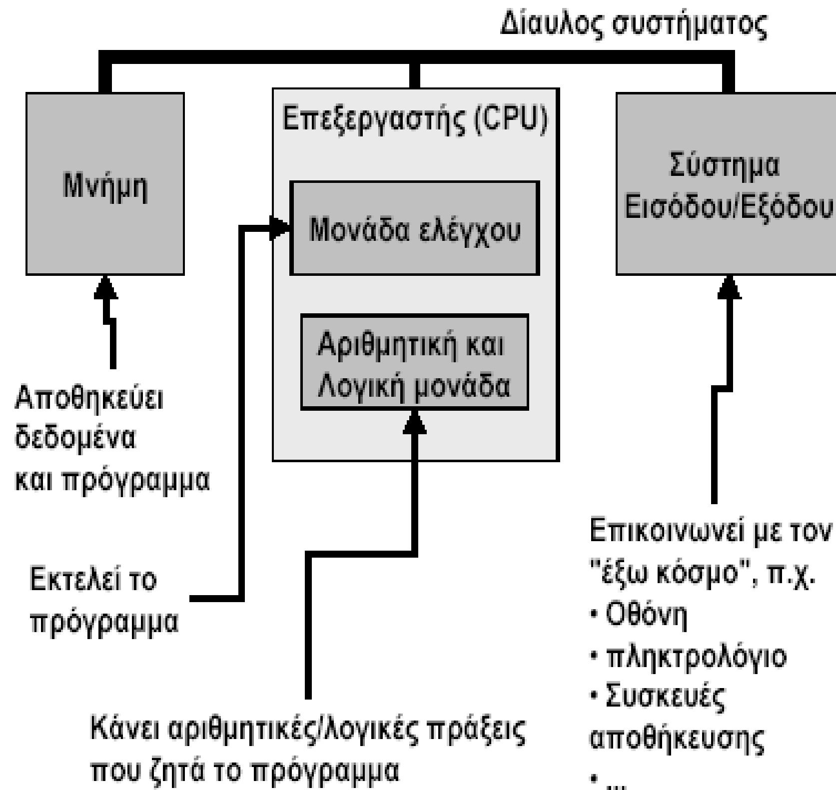
Αρτηρία εισόδου/εξόδου

Μονάδα διασύνδεσης

Θέσεις επέκτασης

Αρτηρία συστήματος

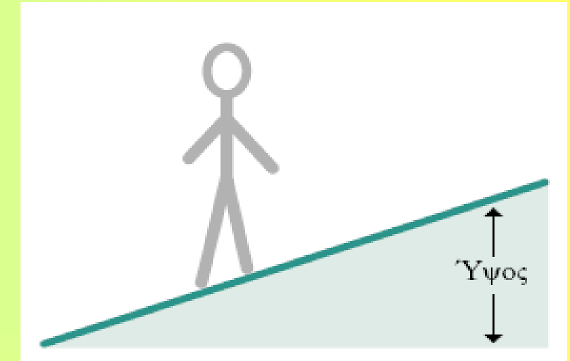
Η αρχιτεκτονική μηχανής Von Neumann



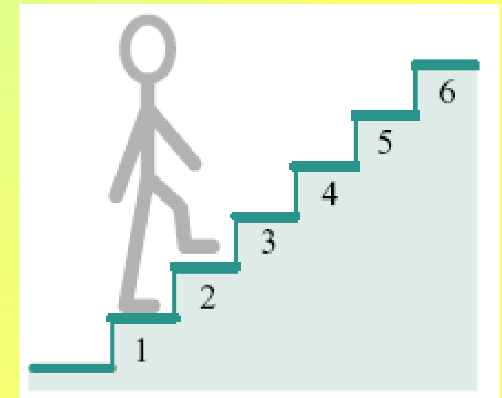
Η μονάδα ελέγχου και η Αριθμητική/Λογική μονάδα βρίσκονται μαζί σε στενή αλληλεξάρτηση και ονομάζονται μαζί "Επεξεργαστής" ή Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit – CPU)

ΕΠΙΠΕΔΑ ΜΕΛΕΤΗΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

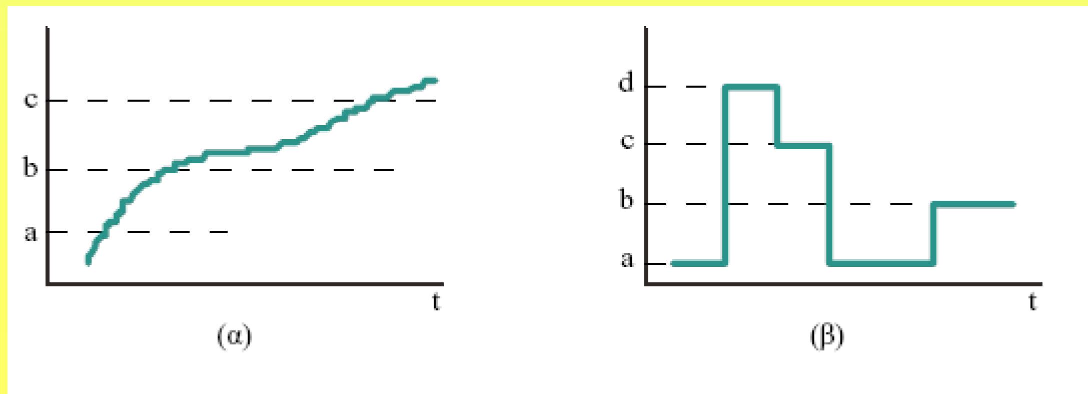
Αναλογικό vs Ψηφιακό



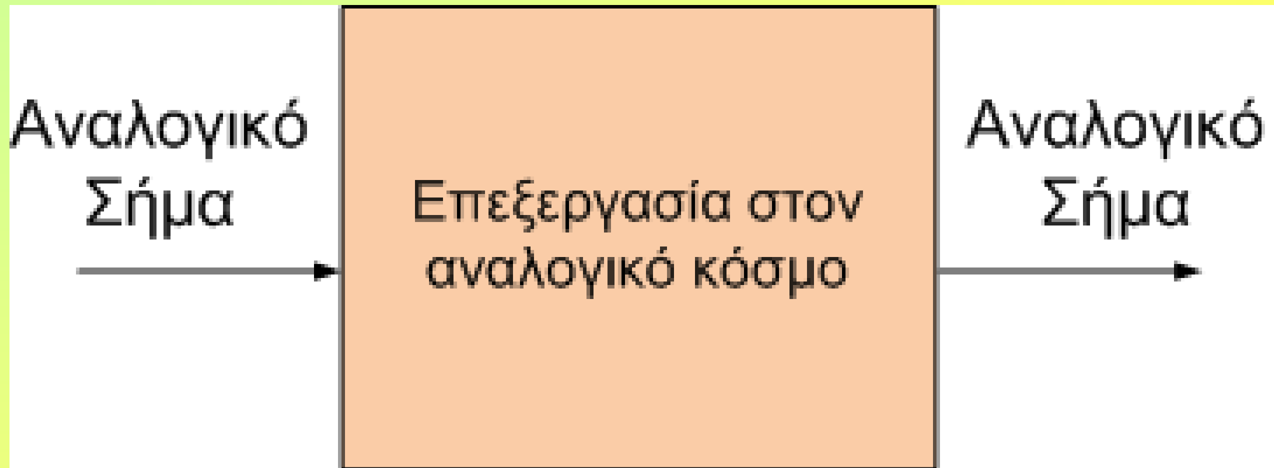
- Αναλογικό είναι ένα μέγεθος (σήμα) όταν μπορεί να πάρει οποιαδήποτε τιμή από ένα συνεχές διάστημα τιμών.
- Η πλειοψηφία των μεγεθών στο κόσμο μας είναι αναλογικά.



- Ψηφιακό (κβαντισμένο) είναι ένα μέγεθος (σήμα) όταν μπορεί να πάρει διακριτές τιμές.
- Φωτεινοί σηματοδότες, διακόπτες είναι παραδείγματα ψηφιακών μεγεθών.



Γιατί να μη γίνεται η επεξεργασία στον αναλογικό κόσμο ?

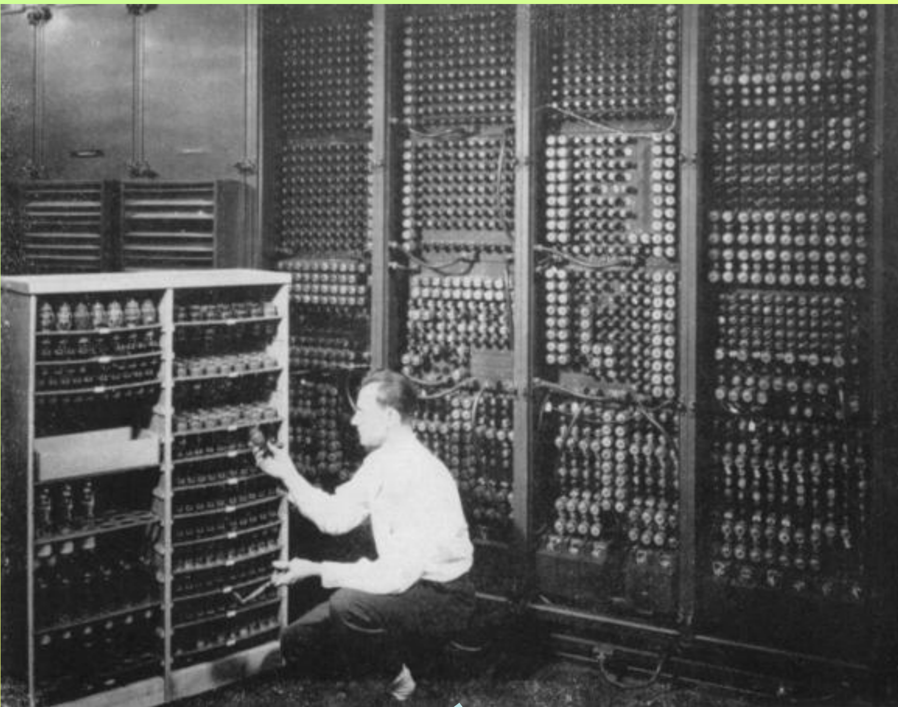


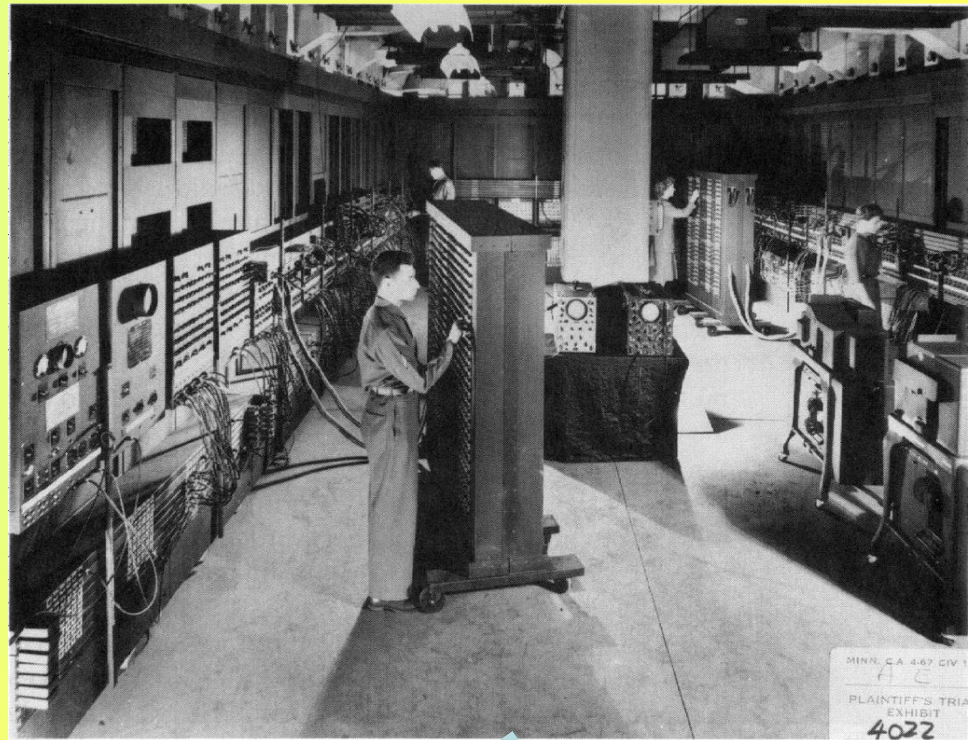
Θα μπορούσε να γίνει αν δεχόμασταν :

- το CD player μας να είναι όσο το αμφιθέατρο
- το CD player μας να κοστίζει όσο ένα ιδιωτικό νησί
- το CD player μας να λειτουργεί μία φορά στις 10.000.000
- το CD player μας να είναι το ίδιο για καμμιά 20αριά χρόνια

Μα κυρίως ...

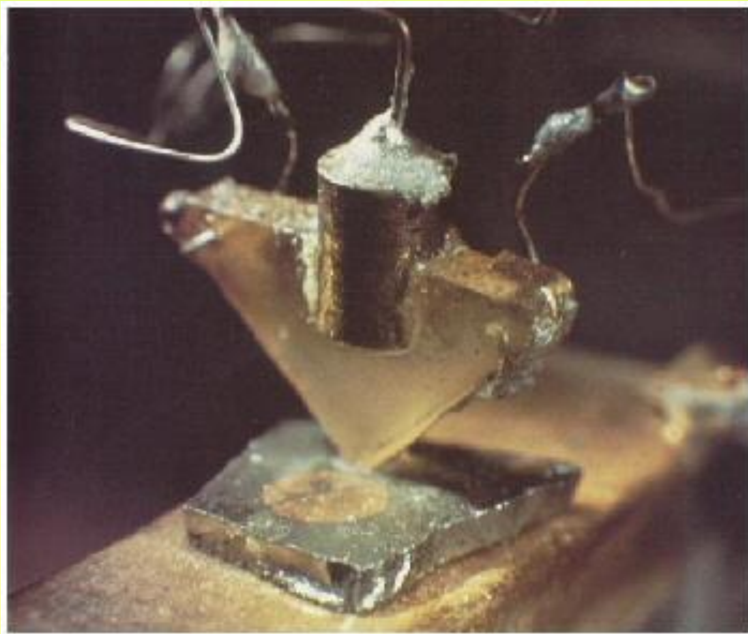
Διαρκώς μειούμενο μέγεθος !

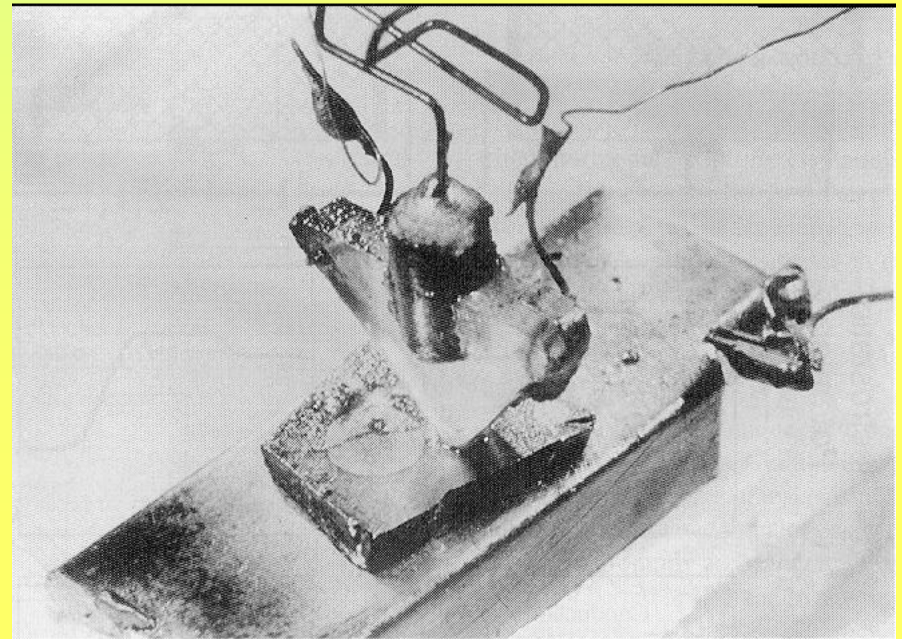




Πως επιτεύχθηκε τόσο δραστική μείωση του μεγέθους ?

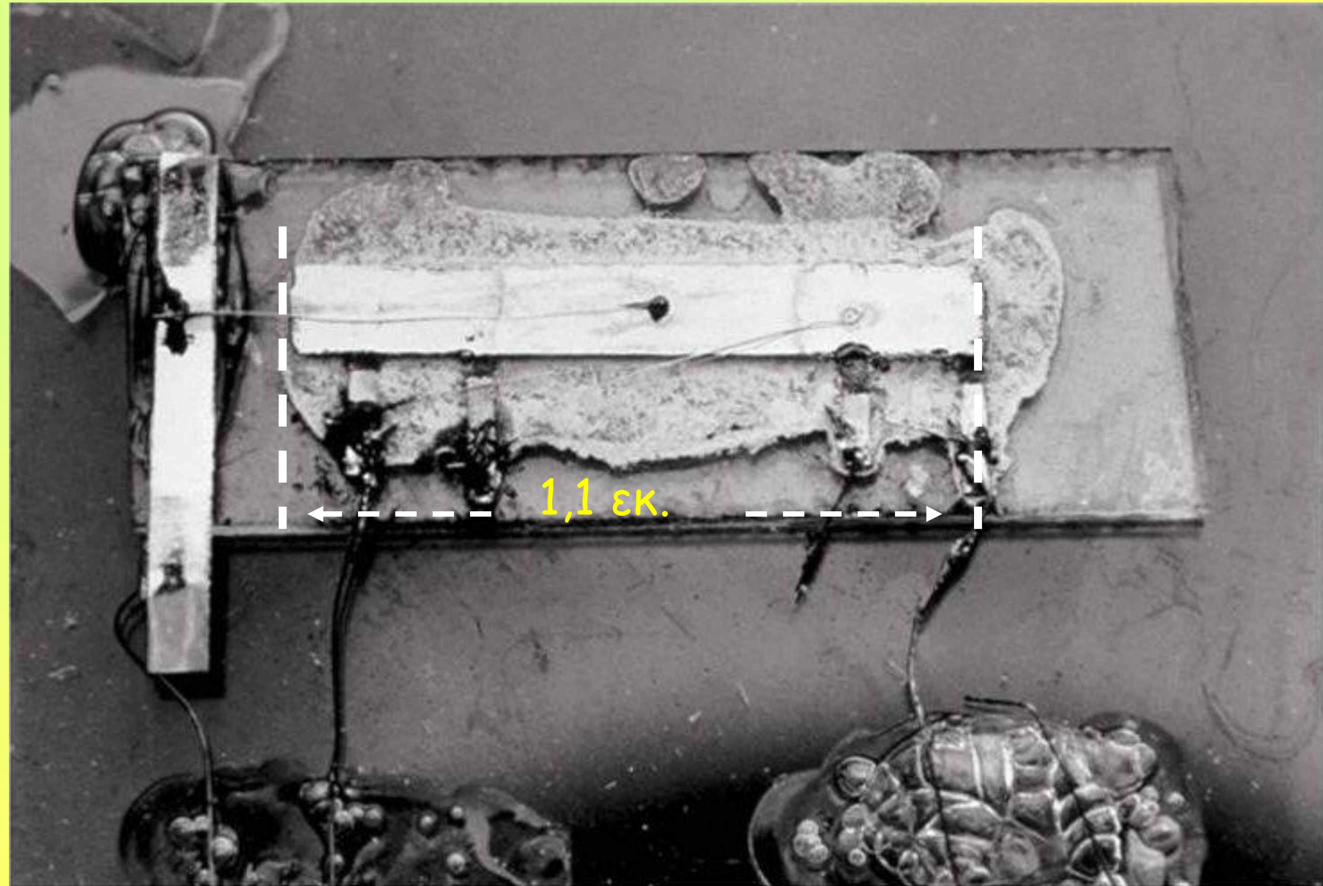
- Το πρώτο μισό του 20ου αιώνα τα ηλεκτρονικά κυκλώματα αποτελούνταν από λυχνίες κενού
  Μεγάλες, ακριβές, ενεργοβόρες, αναξιόπιστες
- 1947: Η εφεύρεση του πρώτου transistor στα Bell Labs από τους John Bardeen και Walter Brattain
  (Nobel Φυσικής το 1956 μαζί με τον William Shockley)





Κατασκευή του transistor πάνω σε ημιαγωγό (Ολοκληρωμένο κύκλωμα – 1958)

Jack Kilby
Texas Instruments
(Nobel Φυσικής,
2000)

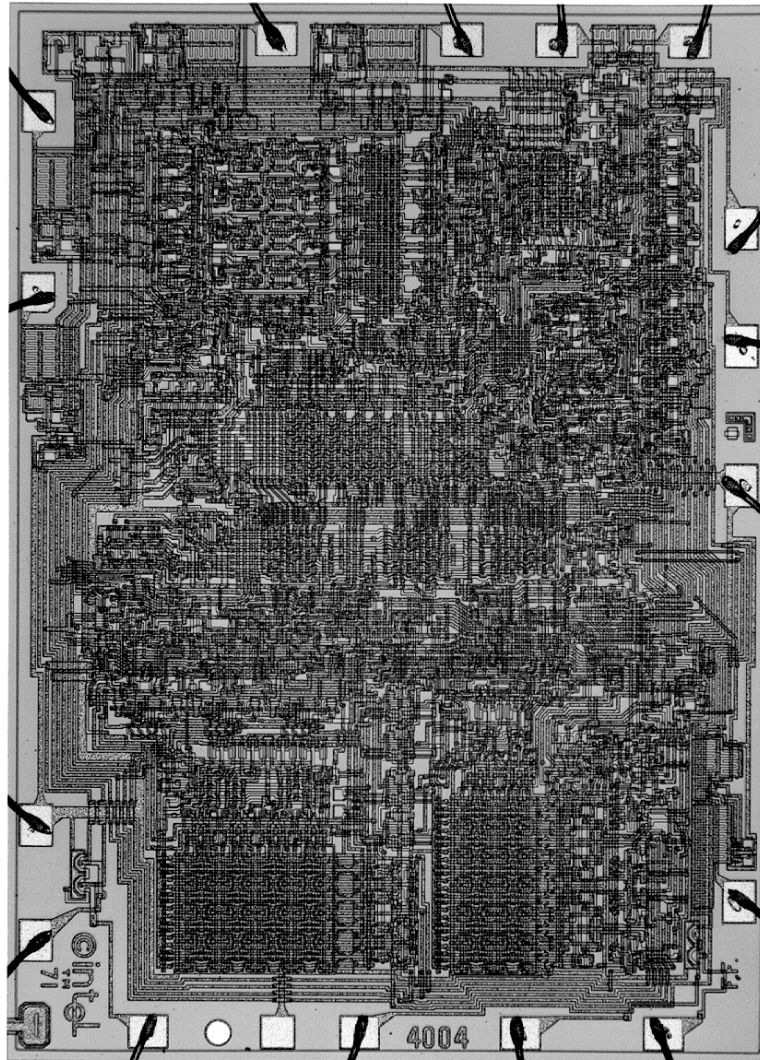


Το παιχνίδι της σμίκρυνσης

- Μικρής κλίμακας ολοκλήρωσης (SSI): > 10 τρανζίστορ
- Μεσαίας κλίμακας ολοκλήρωσης (MSI): > 100 τρανζίστορ
- Υψηλής κλίμακας ολοκλήρωσης (LSI): > 1000 τρανζίστορ
- Πολύ υψηλής κλίμακας ολοκλήρωσης (VLSI): > 10000 τρανζίστορ

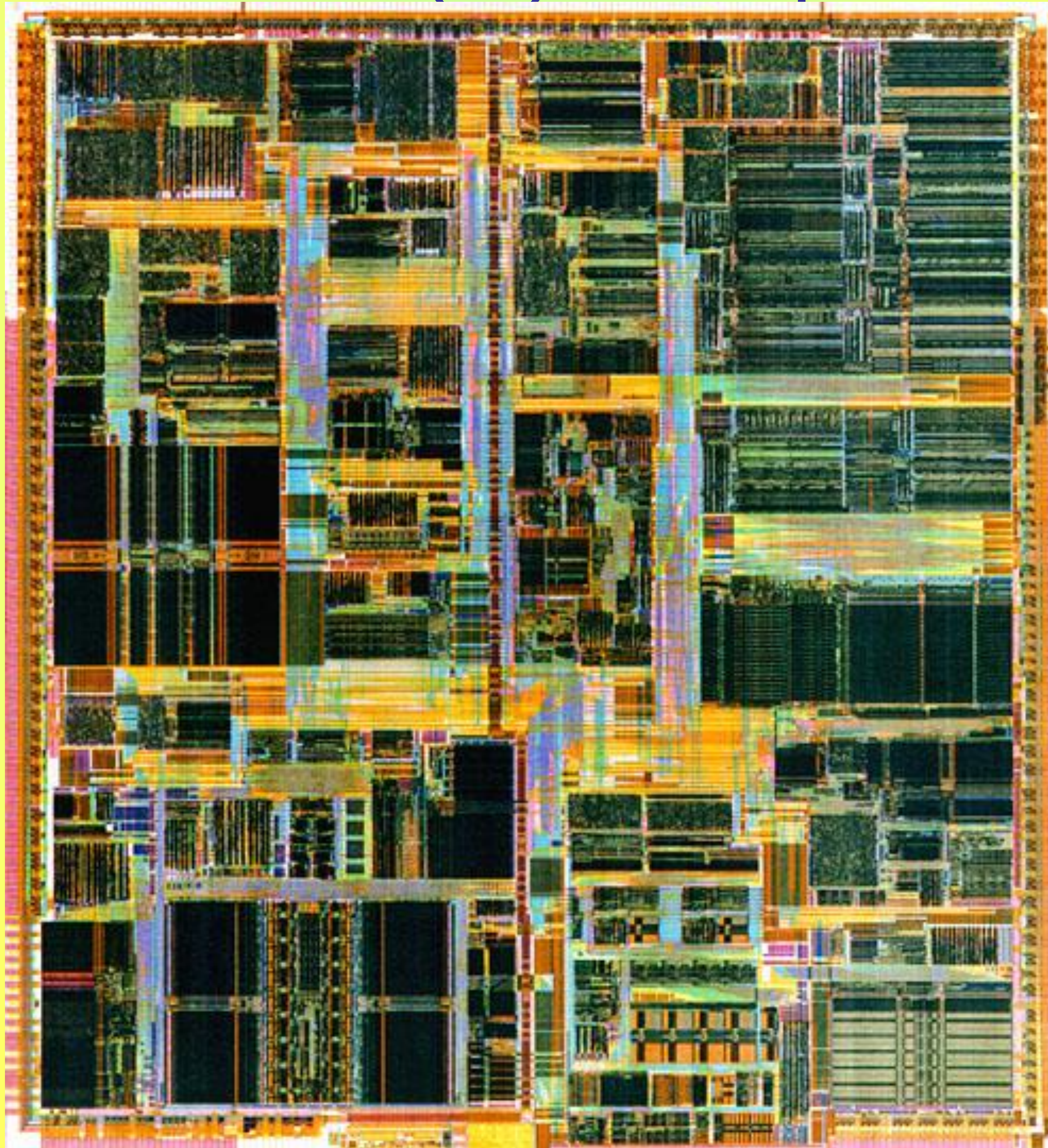
- 2003:
  - Intel Pentium 4 mprocessor (55 εκατομμύρια τρανζίστορ)
  - 512 Mbit DRAM (> 0.5 δις τρανζίστορ)

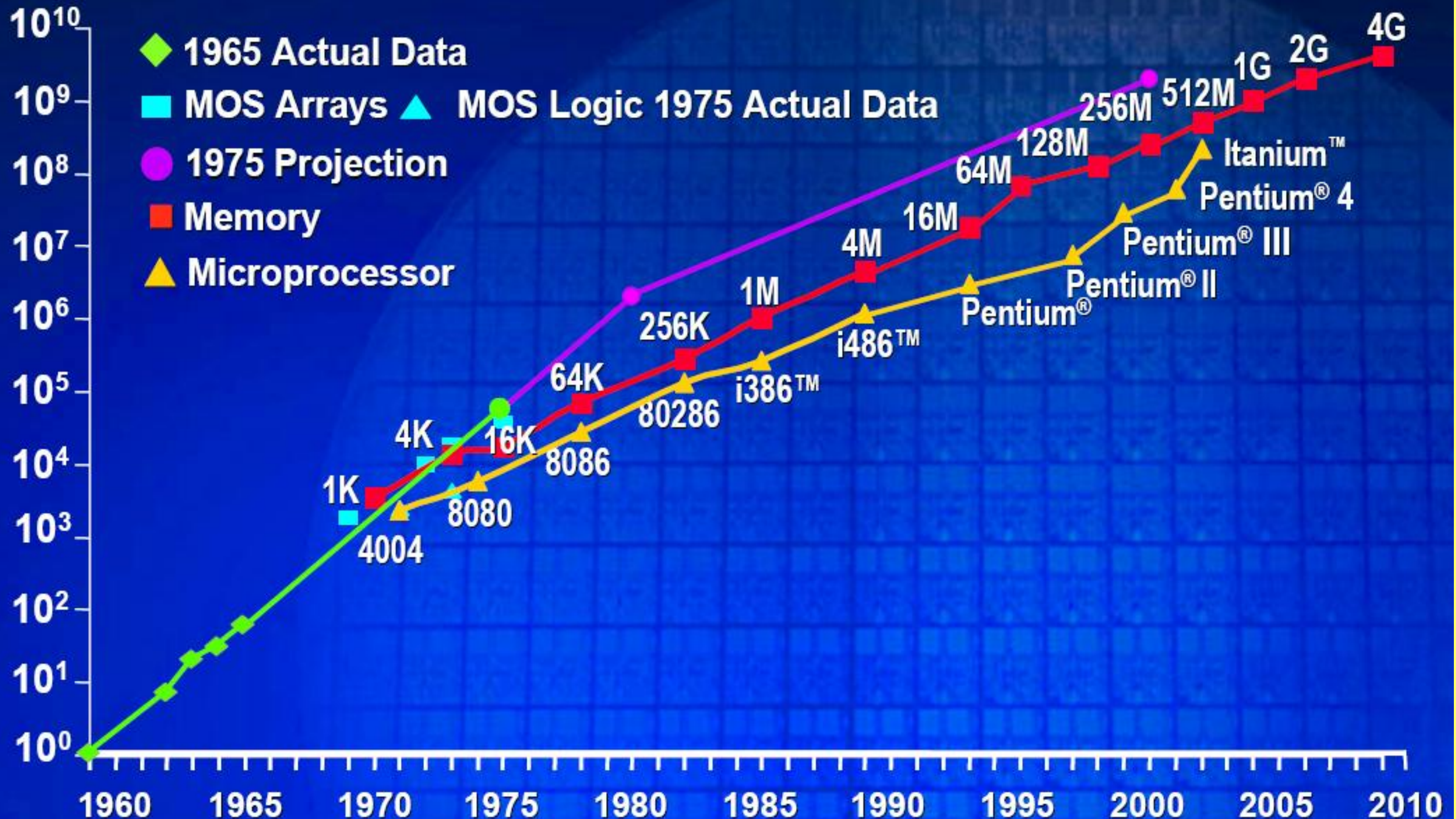
Intel 4004 Micro-Processor



1971
1000 transistors
1 MHz operation

Intel Pentium (IV) microprocessor





Integrated Circuit Complexity
Transistors Per Die
1965 Actual Data
MOS Arrays
MOS Logic 1975 Actual Data
1975 Projection
Memory
Microprocessor
1K
4K
16K
64K
256K
1M
4M
16M
64M
128M
256M
512M
1G
2G
4G
4004
8080
8086
80286
i386™
i486™
Pentium®
Pentium® II
Pentium® III
Pentium® 4
Itanium™
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010

Σημερινή κατάσταση

1. Υπερυπολογιστές (Supercomputers)
2. Μεγάλα Συστήματα (Mainframes)
3. Σταθμοί εργασίας (Workstations)
4. Μικροϋπολογιστές (Microcomputers)
5. Μικροελεγκτές (Microcontrollers)

Υπερυπολογιστές (Supercomputers)

Μεγάλα Συστήματα (Mainframes)

Workstation: Sun Ultra450

microcomputer

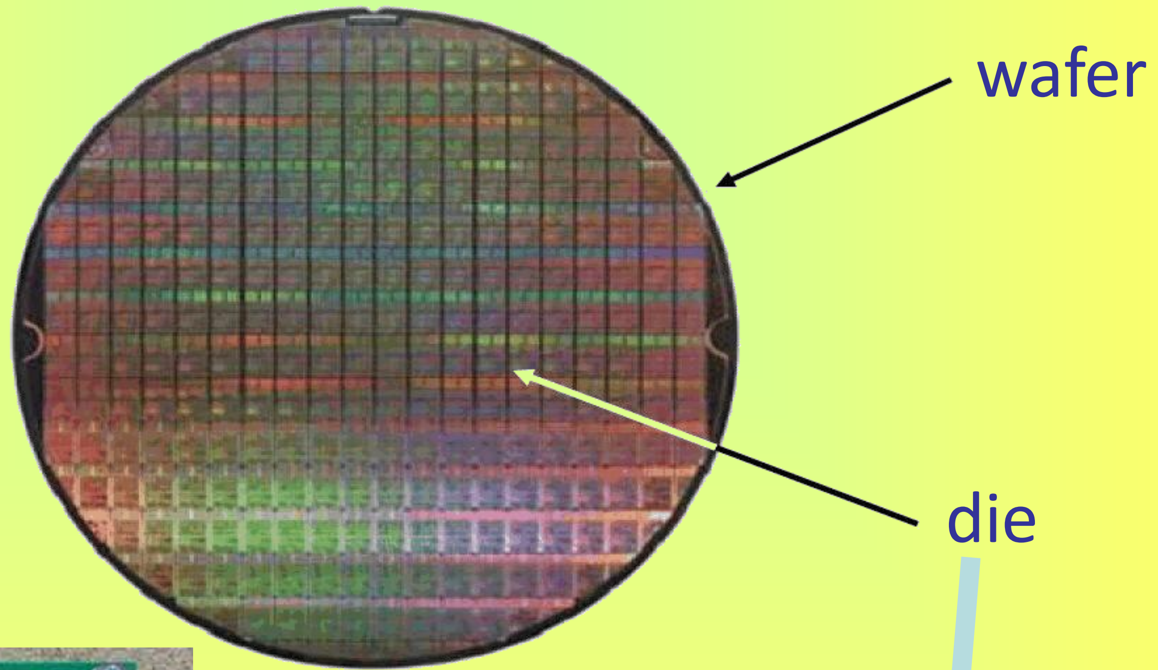
Personal Digital Assistant

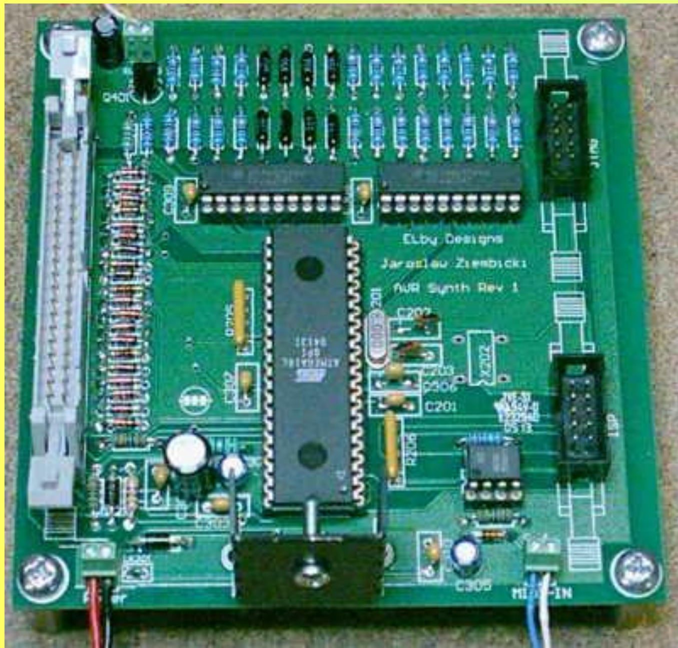
Το μάθημα της Ψηφιακής Σχεδίασης

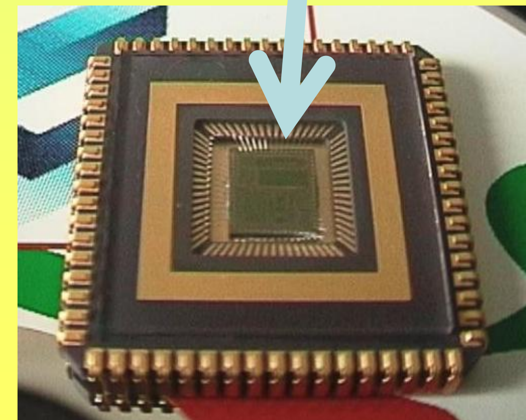
- Στοχεύει να σας μάθει πώς σχεδιάζονται τα Ψηφιακά Συστήματα
- Διάφορες συνιστώσες :
  - Βασικές αρχές (θεωρία)
  - Παραδείγματα (φροντιστήριο)
  - Μικρές εφαρμογές στο πραγματικό κόσμο (εργαστηριακό μάθημα) και πιο μεγάλες και πιο ενδιαφέρουσες εφαρμογές στον ιδεατό (εξομοιούμενο) κόσμο (ασκήσεις)

ΕΠΙΠΕΔΑ ΜΕΛΕΤΗΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

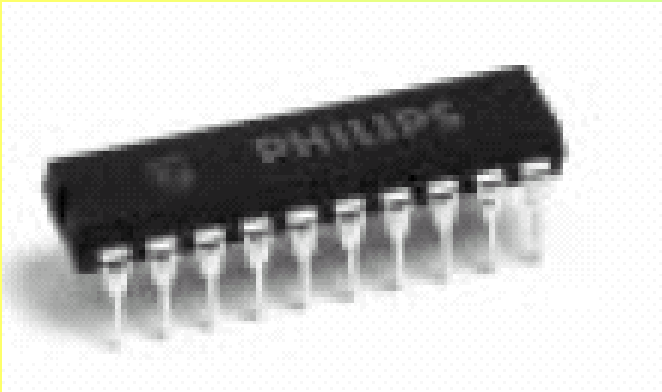
Οι πολλές μορφές ενός ψηφιακού κυκλώματος

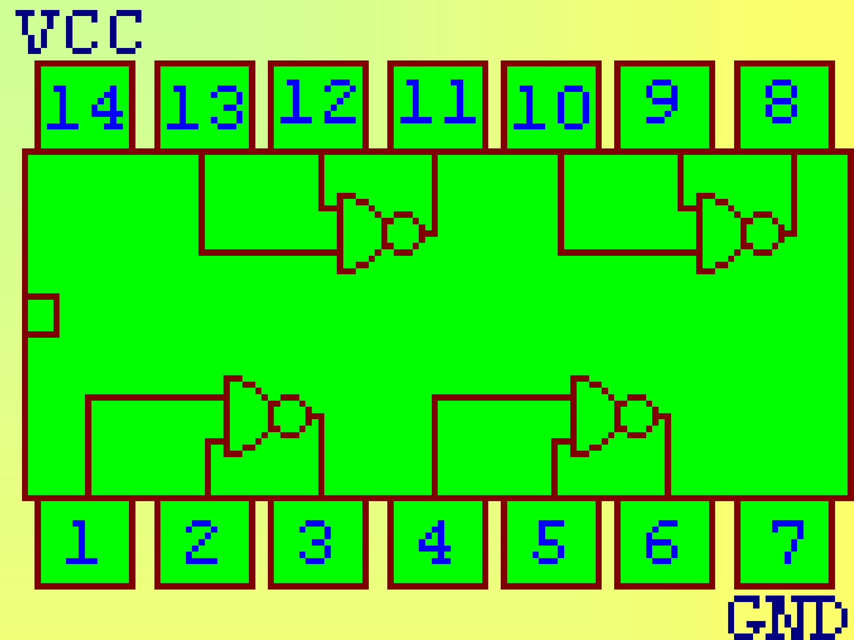






Και ο πραγματικός κόσμος που αντιπροσωπεύει





Κάθε πύλη εκτελεί μια "λογική συνάρτηση", συνάρτηση δηλαδή πάνω σε δυαδικές μεταβλητές

Αναπαράσταση πληροφορίας με δυαδικά σήματα

- Εστω ότι έχω ένα διακόπτη που μπορεί να είναι κλειστός ή ανοικτός.
- Μπορώ να κάνω την αντιστοιχία :

  0 -> ανοικτός

  1 -> κλειστός
- Μου αρκεί ένα δυαδικό σήμα (ισοδύναμα μια δυαδική μεταβλητή ή ένα bit) για να περιγράψω τη κατάστασή του.

- Πόσα bits χρειάζονται για τη περιγραφή της κατάστασης 2 διακοπτών ?
- Πόσα bits για τη περιγραφή της κατάστασης 10 διακοπτών ?

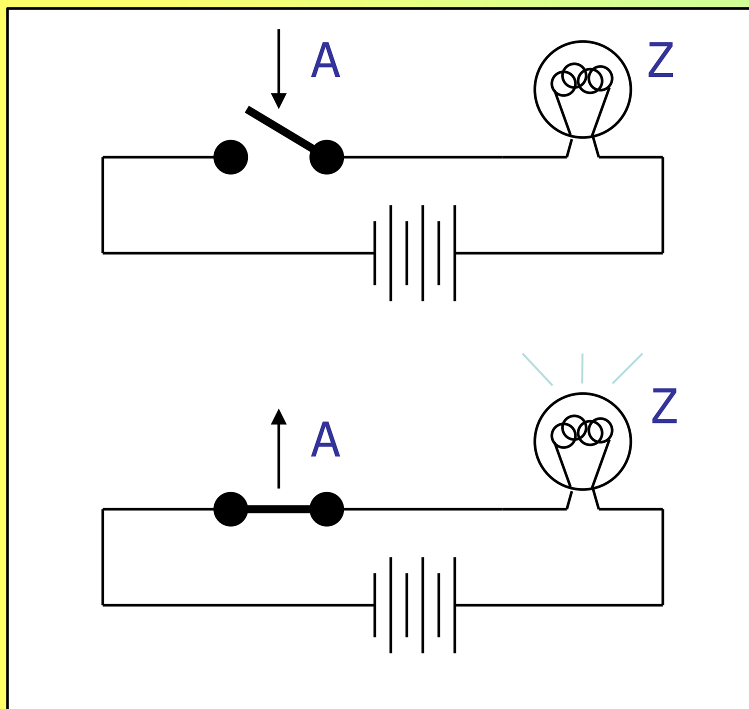
Με n δυαδικά ψηφία μπορώ να αναπαραστήσω 2^n διακριτά πράγματα

m διακριτά πράγματα χρειάζονται κατ' ελάχιστο $\lceil \log_2 m \rceil$ δυαδικά ψηφία

Ποια αναπαράσταση είναι καλύτερη ?

- **Εκείνη που χρησιμοποιεί τα λιγότερα δυαδικά σήματα**
- **Εκείνη που μου επιτρέπει να κάνω εύκολα την επεξεργασία της πληροφορίας**
- **Πως θα αναπαριστούσατε μια ημέρα από τη 1/1/0001 και μετά ?**
  - **20 bit τα λιγότερα !**
  - **Ημέρα(365) Μήνας (12), Χρόνος (2000+) => (5 + 4 + 12) => 21**
  - **Ημέρα, Μήνας, Χρόνος, Μέρα της εβδομάδας (7) => (5 + 4 + 12 + 3) => 24**

Λογικές (Δυαδικές) Συναρτήσεις

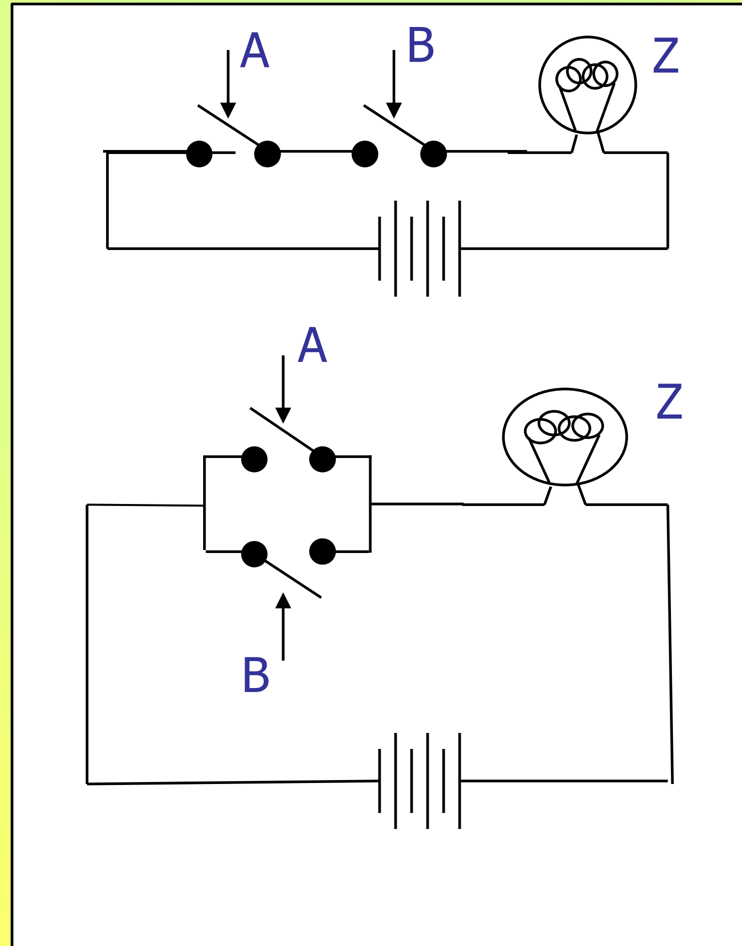


- Υποθέστε
  - τη δυαδική μεταβλητή A
    - A=0 => διακόπτης ανοικτός
    - A=1=> διακόπτης κλειστός
  - και τη Z
    - Z=0 => Οχι φώς
    - Z=1 => Φως

Η Z είναι ανεξάρτητη από τη A ή όχι ?

Η Z είναι μια συνάρτηση της A, ισχύει Z = f(A) = A!

Απλές λογικές συναρτήσεις 2 δυαδικών μεταβλητών

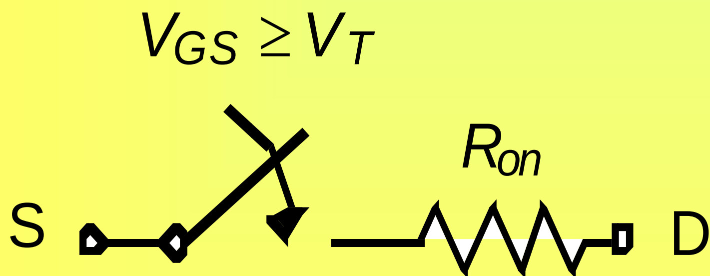


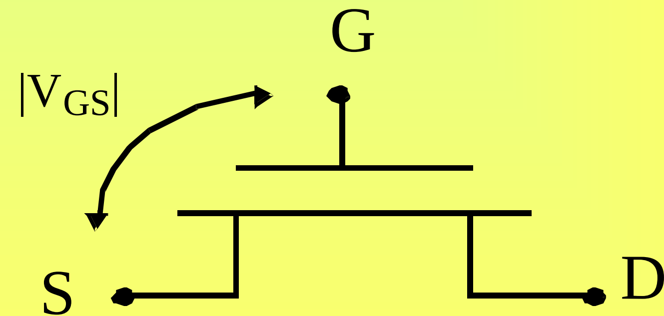
Οι συναρτήσεις αυτές ονομάζονται και λογικές γιατί τις χρησιμοποιούμε στη λογική των προτάσεων που συντάσσουμε καθημερινά.

Ποιοι είναι οι διακόπτες στον κόσμο της Ηλεκτρονικής;

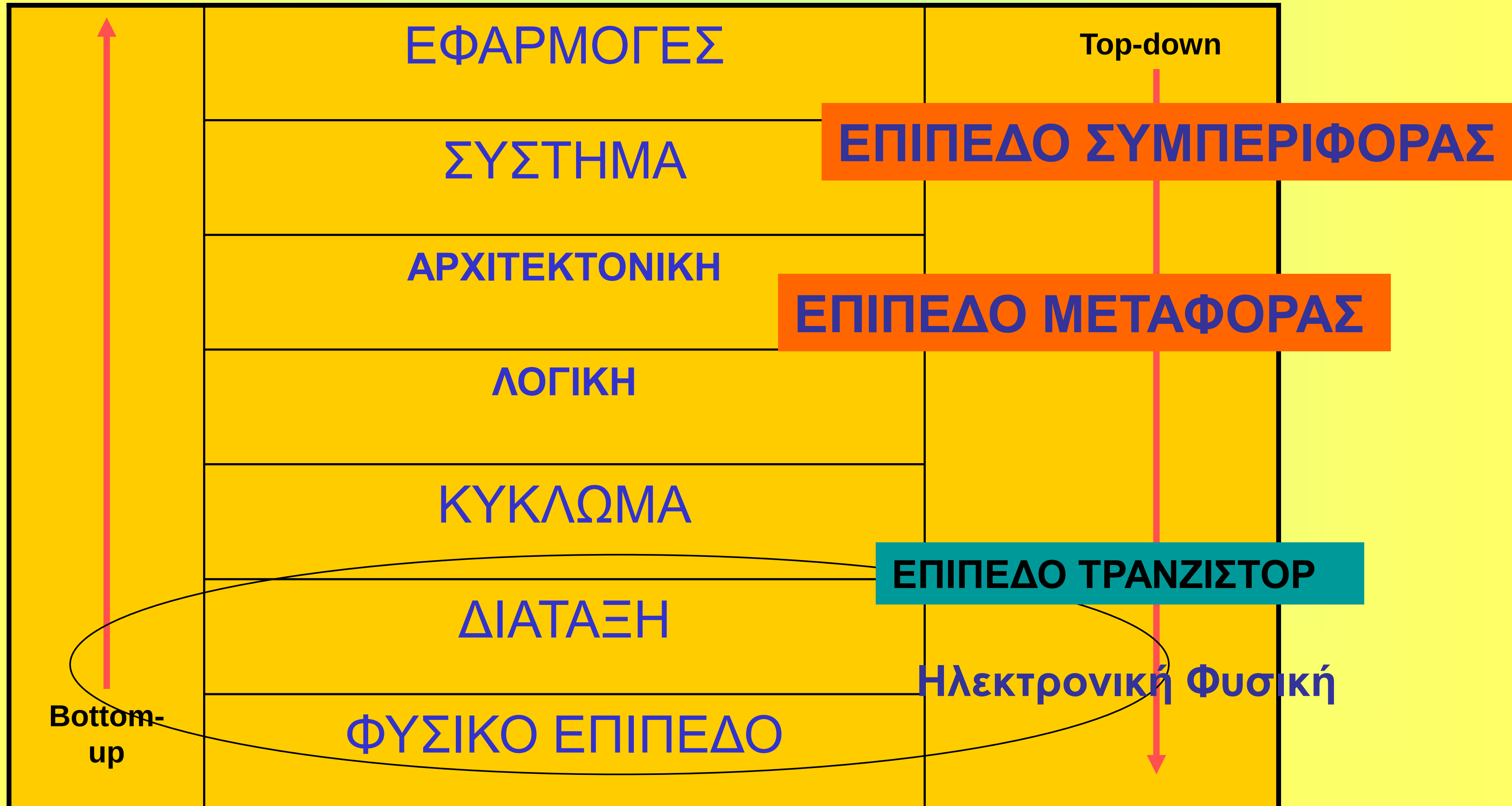
- Απάντηση: Τα τρανζίστορ

Ένας Διακόπτης! ⟷ Ένα Τρανζίστορ **MOS**

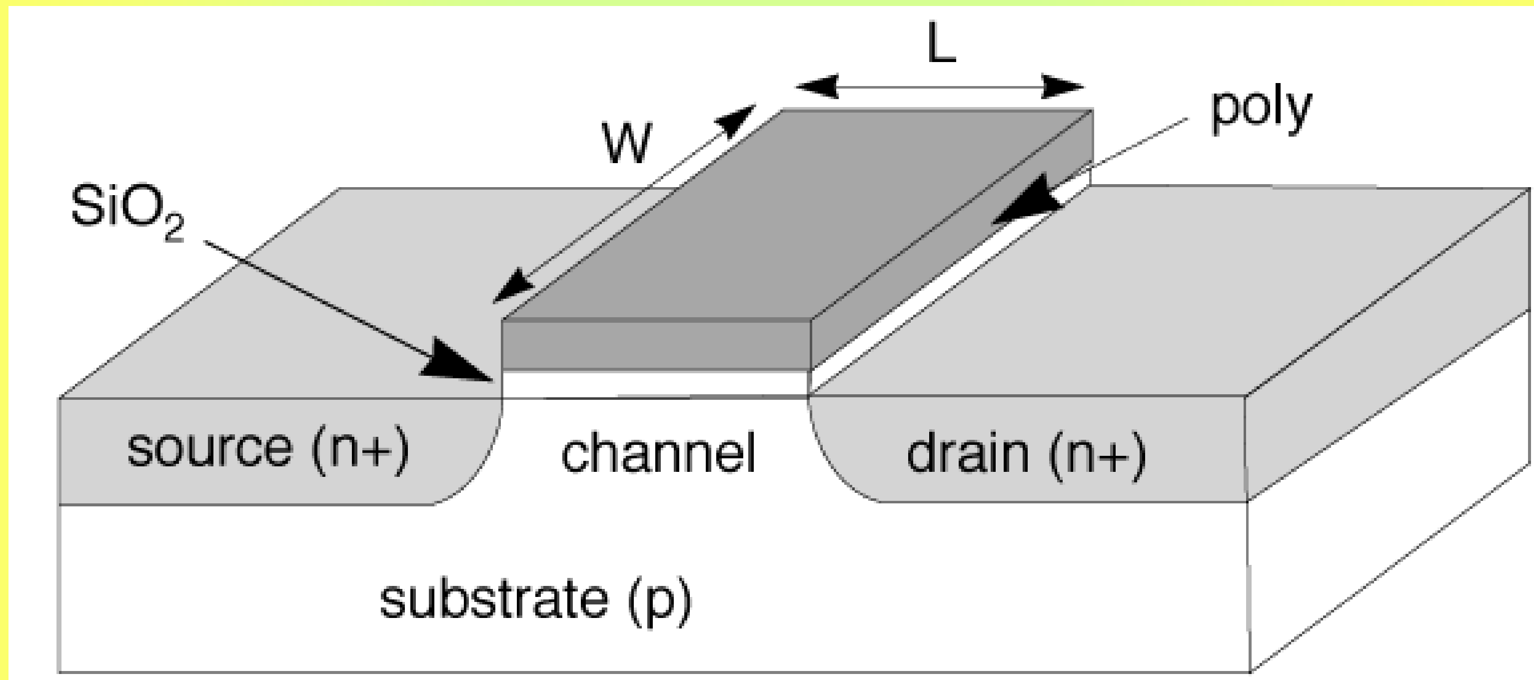




ΕΠΙΠΕΔΑ ΜΕΛΕΤΗΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ



Τρανζίστορ MOSFET



Ηλεκτρικές Ιδιότητες Των Υλικών.

Τα υλικά μπορούν να ταξινομηθούν με βάση την ηλεκτρική τους αγωγιμότητα σε:

α) Αγωγούς, που είναι κυρίως τα Μέταλλα αλλά και τα κράματά τους. Στους αγωγούς, φορείς του ρεύματος είναι τα ελεύθερα ηλεκτρόνια που κινούνται στο κρυσταλλικό πλέγμα και για τον λόγο αυτό τα μέταλλα παρουσιάζουν ηλεκτρική αγωγιμότητα όταν εφαρμοστεί σε αυτά ένα ηλεκτρικό πεδίο. Στα ελεύθερα ηλεκτρόνια οφείλεται και η μεταλλική λάμψη των μετάλλων.

β) Μονωτές ή διηλεκτρικά, στους οποίους όλοι οι φορείς (ηλεκτρόνια) είναι πλήρως και σταθερά δεσμευμένα στα άτομα του κρυσταλλικού πλέγματος και μια συνηθισμένη προσφορά ενέργειας δεν μπορεί να δημιουργήσει ελεύθερα ηλεκτρόνια, άρα δεν παρουσιάζουν ηλεκτρική αγωγιμότητα.

γ) Ημιαγωγούς, όπως το πυρίτιο, που άγουν το ρεύμα κάτω από ορισμένες συνθήκες (λ.χ. σχετικά αυξημένη θερμοκρασία, πρόσπτωση φωτός κ.λ.π.) και για τον λόγο αυτό καλούνται ημιαγωγοί.

Στοιχειώδης Φυσική των Ημιαγωγών.

•Ένας ημιαγωγός εμφανίζει πολύ μικρή ηλεκτρική αγωγιμότητα. Στο κρυσταλλικό πλέγμα του ημιαγωγού, μπορεί να υπάρξει ένα ηλεκτρόνιο (e) που δεν είναι στη θέση του, γιατί λόγω θερμικής κινητικής ενέργειας ή από ένα φωτόνιο, ελευθερώθηκε από τον ομοιοπολικό δεσμό σθένους αφήνοντας τη θέση αυτή κενή. ***Την κενή αυτή θέση (έλλειψη ηλεκτρονίου) την ονομάζουμε οπή. Συμπεριφέρεται σαν θετικό φορτίο και δηλώνεται με το p (από το positive).***

→Το ελεύθερο ηλεκτρόνιο μπορεί να κινηθεί μέσα στον κρύσταλλο, συμβάλλοντας στην αγωγιμότητα τού κρυστάλλου.

→Η οπή μπορεί να συμπληρωθεί εύκολα από ένα γειτονικό ηλεκτρόνιο γιατί τα ηλεκτρόνια ταλαντεύονται γύρω από τη θέση ισορροπίας τους κι ένα από αυτά μπορεί να εγκατασταθεί στη θέση τής οπής συμπληρώνοντας τον ατελή δεσμό. Τότε βέβαια μια άλλη οπή θα εμφανισθεί στη δική του προηγούμενη θέση και είναι σαν να μετακινήθηκε η αρχική οπή.

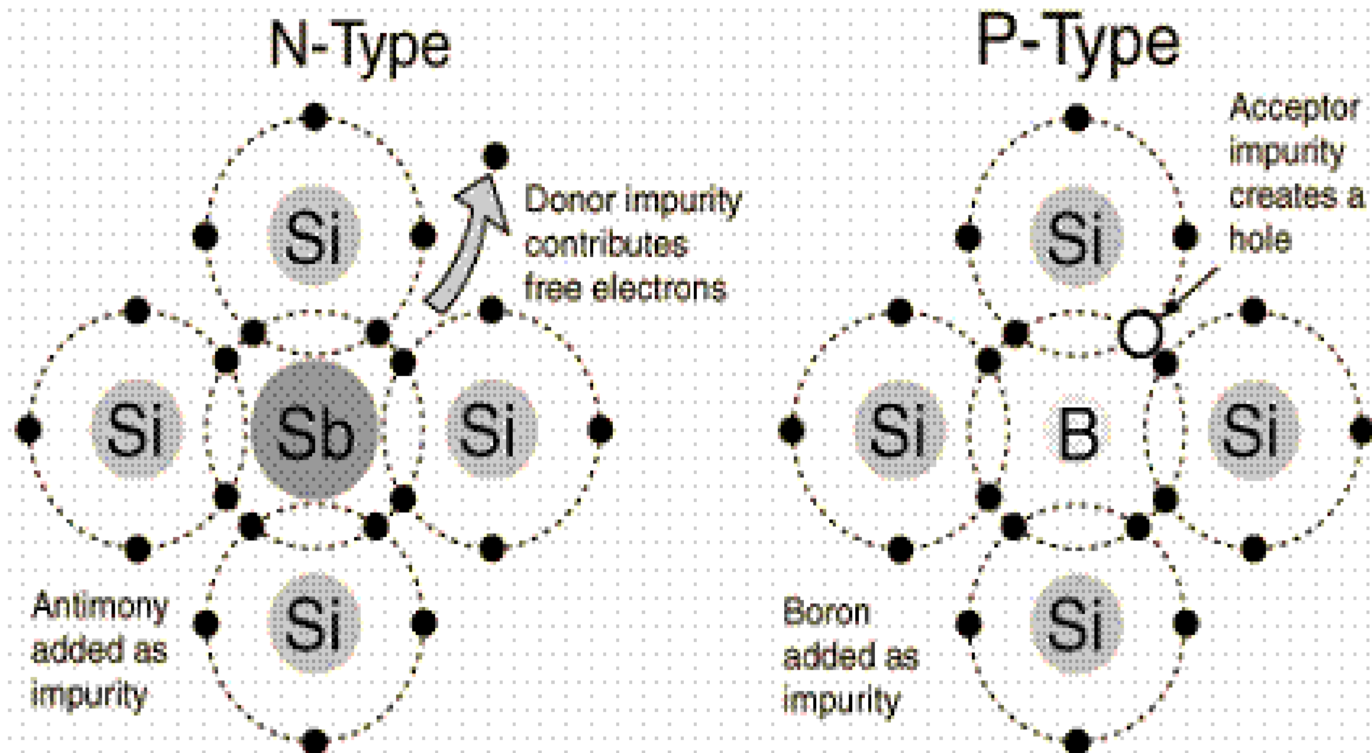
•Αν εφαρμοσθεί ηλεκτρικό πεδίο *E* θα έχουμε μετακίνηση φορτίου για δύο λόγους: α) Από την προσανατολισμένη κίνηση των ελευθέρων ηλεκτρονίων και β) από την προσανατολισμένη μετατόπιση των οπών. Οι δύο μετακινήσεις φορτίων γίνονται κατ' αντίθετη φορά και επομένως τόσο οι οπές όσο και τα ελεύθερα ηλεκτρόνια συμβάλλουν στην αγωγιμότητα του κρυστάλλου κατά την ίδια φορά. Επειδή η αγωγιμότητά τους είναι από την φύση τους παρούσα, οι ημιαγωγοί αυτοί ονομάζονται «ενδογενείς» (intrinsic semiconductors).

Διαδικασία εμπλουτισμού ή νόθευσης (doping).

❖Εισαγωγή ενός πεντασθενούς ή ενός τρισθενούς στοιχείου στον κρύσταλλο του καθαρού ημιαγωγού. Δημιουργούνται οι λεγόμενοι ***εξωγενείς ημιαγωγοί***.

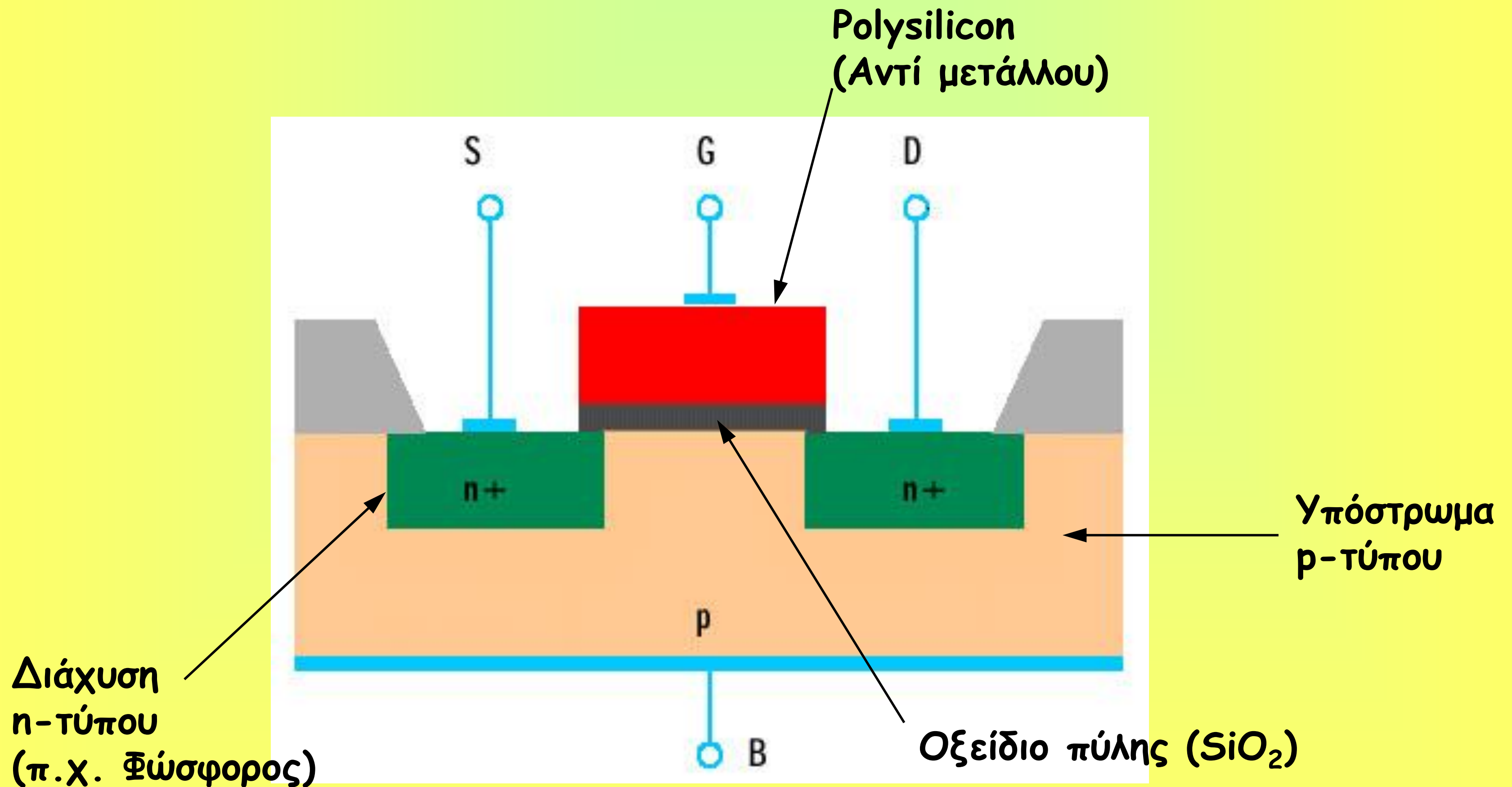
❖Όταν εισαχθούν **πεντασθενή** στοιχεία στον κρύσταλλο του καθαρού ημιαγωγού όπως Φωσφόρος (P) ή Αρσενικό (As). Η διαδικασία εμπλουτισμού δημιουργεί τους λεγόμενους ημιαγωγούς ***τύπου-n*** (από το αγγλικό negative). Τότε τα τέσσερα ηλεκτρόνια σθένους του πεντασθενούς στοιχείου δημιουργούν ομοιοπολικούς δεσμούς με τα τέσσερα του ημιαγωγού (π.χ. Si) και το πέμπτο μένει ελεύθερο να κυκλοφορεί στο κρυσταλλικό πλέγμα του ημιαγωγού, «δίνοντας» έτσι έναν παραπάνω φορέα ηλεκτρικού αρνητικού φορτίου. Τα άτομα των προσμίξεων (impurities) αυτών ονομάζονται «Δότες» (donors).

❖Αν εισαχθούν προσμίξεις από ένα **τρισθενές** στοιχείο όπως το Βόριο (B), το Αλουμίνιο (Al) ή το Γάλλιο (Ga), τότε δημιουργούνται ημιαγωγοί ***τύπου-p*** (από το αγγλικό positive). Στην περίπτωση αυτή, δημιουργούνται οπές αφού τώρα ένα ηλεκτρόνιο λείπει από τον τετραπλό δεσμό, αφήνοντας ένα κενό, δηλαδή έλλειψη αρνητικού φορτίου, σε θέση που περιμένει να «δεχτεί» ένα τέτοιο φορτίο. Τα άτομα των προσμίξεων (impurities) αυτών ονομάζονται «Αποδέκτες» (acceptors).

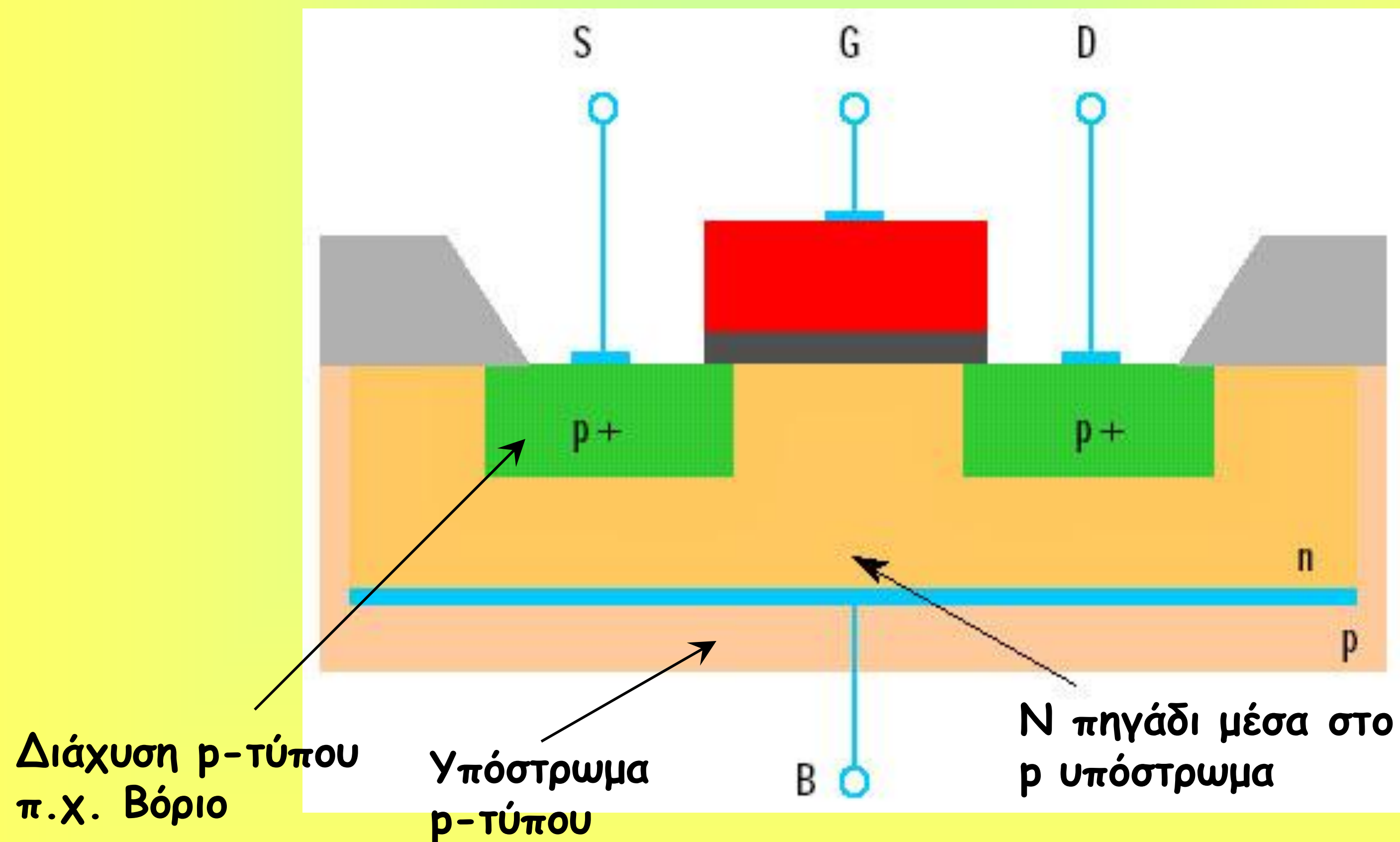


Σχήμα 1. Δημιουργία ημιαγωγών τύπου-n (α) και τύπου-p (β) σε κρύσταλλο πυριτίου.

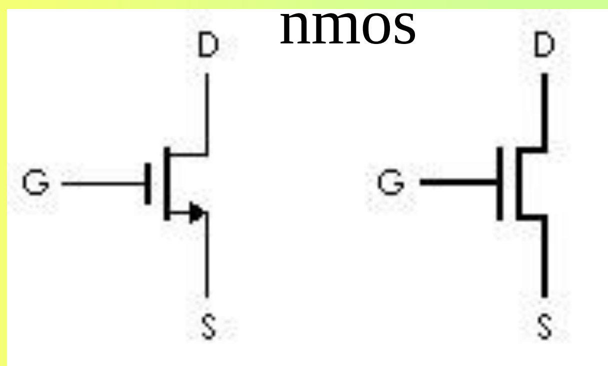
Τρανζίστορ nMOS

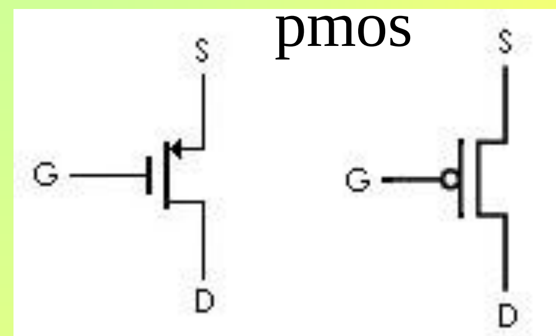


Τρανζίστορ pMOS



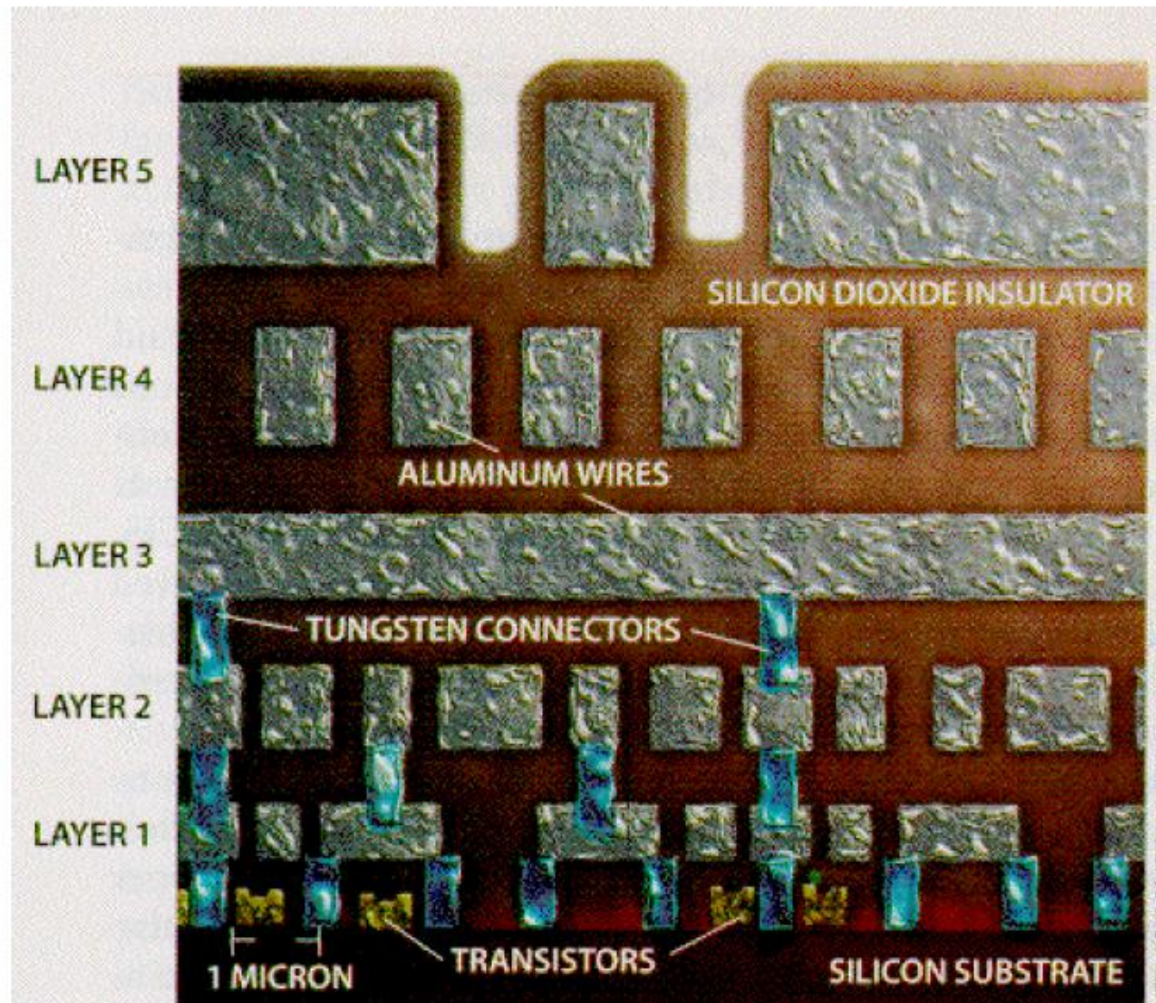
Κυκλωματικά σύμβολα – Καταστάσεις Λειτουργίας





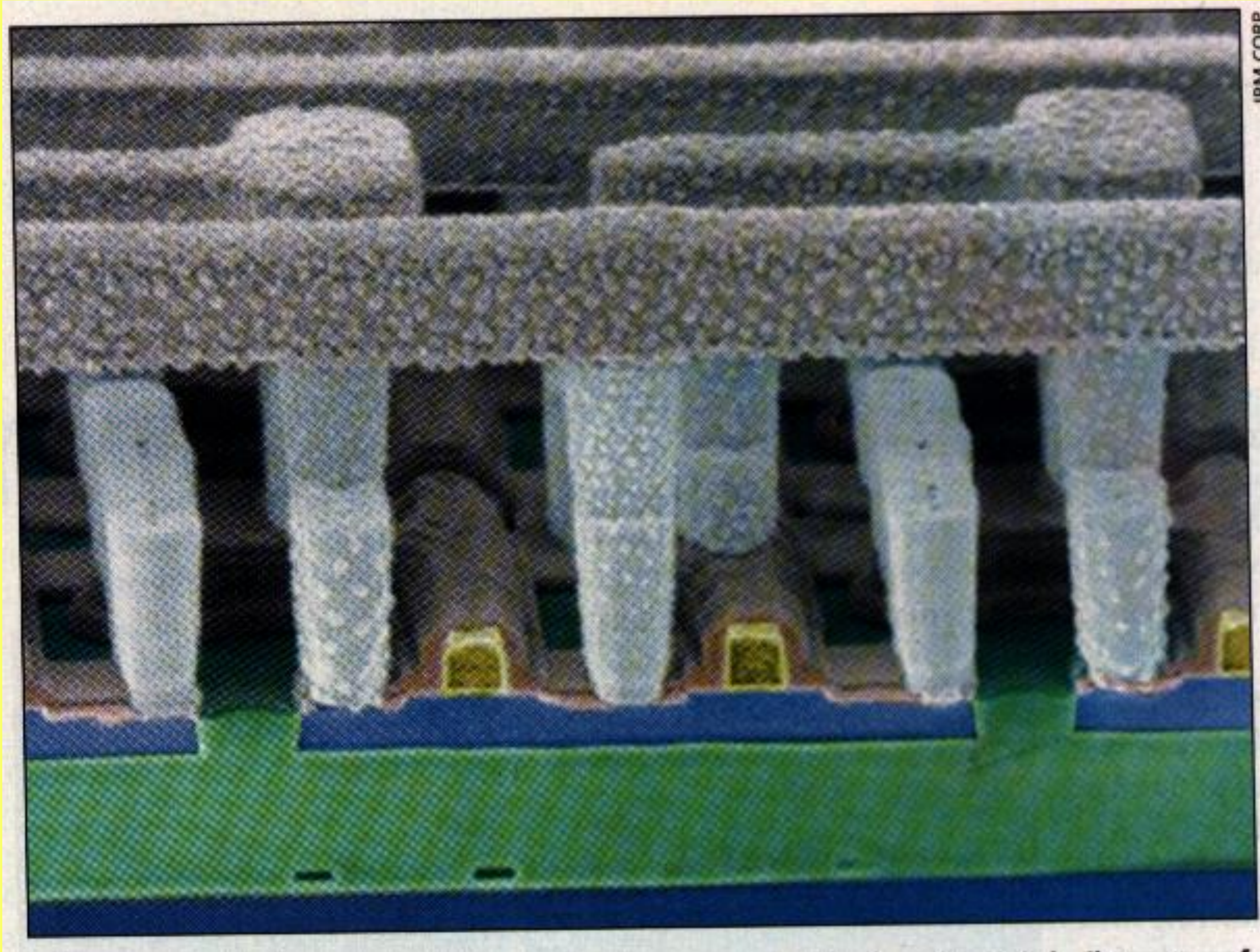
Το τρανζίστορ MOSFET έχει τρείς περιοχές λειτουργίας:

1. Περιοχή αποκοπής
2. Γραμμική περιοχή
3. Περιοχή Κορεσμού



5-layer cross-section of chip

ΜΙΚΡΟΣΚΟΠΙΚΗ ΟΨΗ ΔΙΑΤΑΞΗΣ CMOS

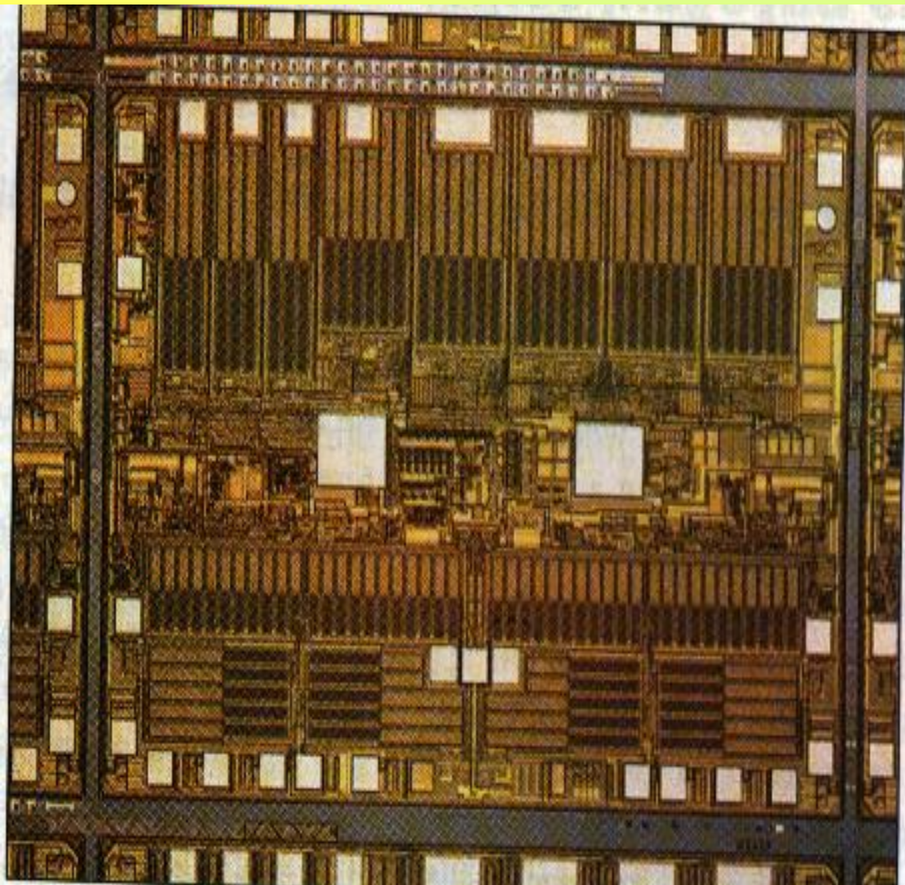


ΕΓΚΑΤΑΣΤΑΣΕΙΣ ΓΙΑ ΤΗΝ ΠΑΡΑΓΩΓΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ

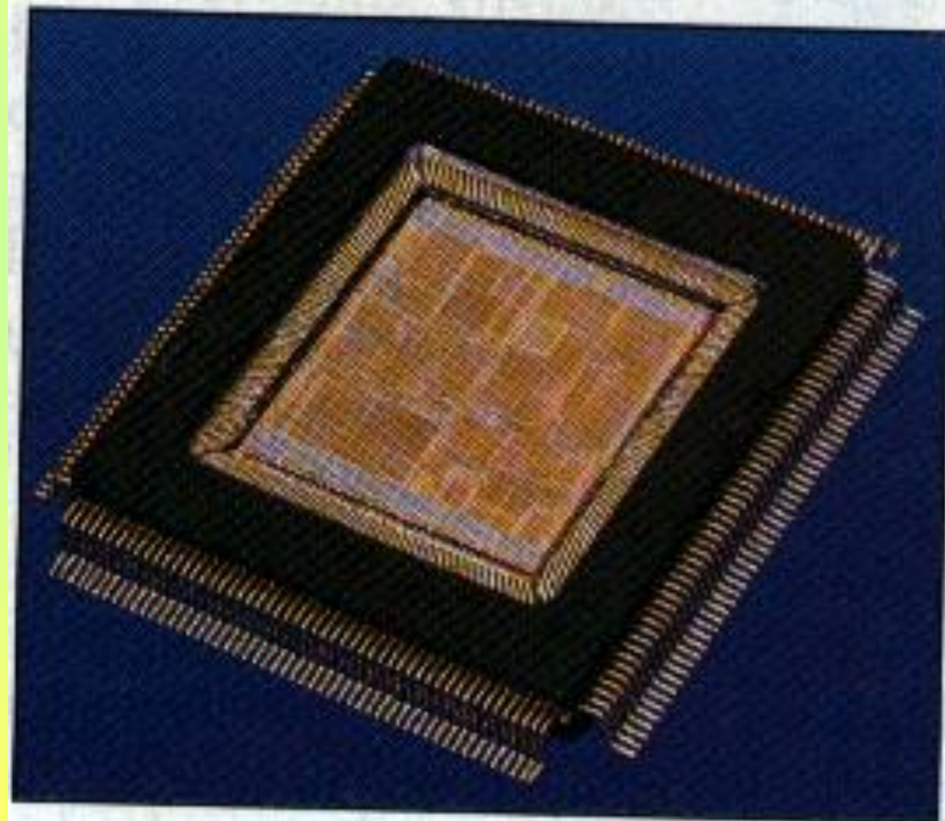
CLEAN ROOM

Σύγχρονος μικροεπεξεργαστής 32-bit

Μικρο-Κύκλωμα



Πλήρες Ολοκληρωμένο.



ΗΛΕΚΤΡΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ

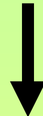
•**Παλαιότερα υπήρχαν οι ηλεκτρονικες δίοδοι, τρίοδοι κλπ λυχνίες κενού. Σήμερα βρίσκονται είτε στα μουσεία είτε σε ορισμένες συσκευές ισχύος.**

•Αντικαταστάθηκαν από τις **Διατάξεις Στερεάς Κατάστασης** (Solid State Devices). Η λειτουργία τους βασίζεται στις ιδιότητες των **ΗΜΙΑΓΩΓΩΝ**.

•Κατασκευάζονται **ΔΙΑΚΡΙΤΕΣ ΔΙΑΤΑΞΕΙΣ ή** και **ΟΛΟΚΛΗΡΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ (MICROCHIPS).**

+

ΠΑΘΗΤΙΚΑ ΣΤΟΙΧΕΙΑ
(R, L, C)



ΗΛΕΚΤΡΟΝΙΚΑ ΚΥΚΛΩΜΑΤΑ
(ΔΙΑΚΡΙΤΑ ή ΟΛΟΚΛΗΡΩΜΕΝΑ)

ΒΑΣΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΣΤΕΡΕΑΣ ΚΑΤΑΣΤΑΣΗΣ:

1. ΔΙΟΔΟΙ
2. ΔΙΠΟΛΙΚΑ ΤΡΑΝΖΙΣΤΟΡ ΕΠΑΦΗΣ
3. ΤΡΑΝΖΙΣΤΟΡ ΕΠΙΔΡΑΣΗΣ ΠΕΔΙΟΥ(FET).
4. ΟΠΤΟΗΛΕΚΤΡΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ (LED, LASERS κλπ)

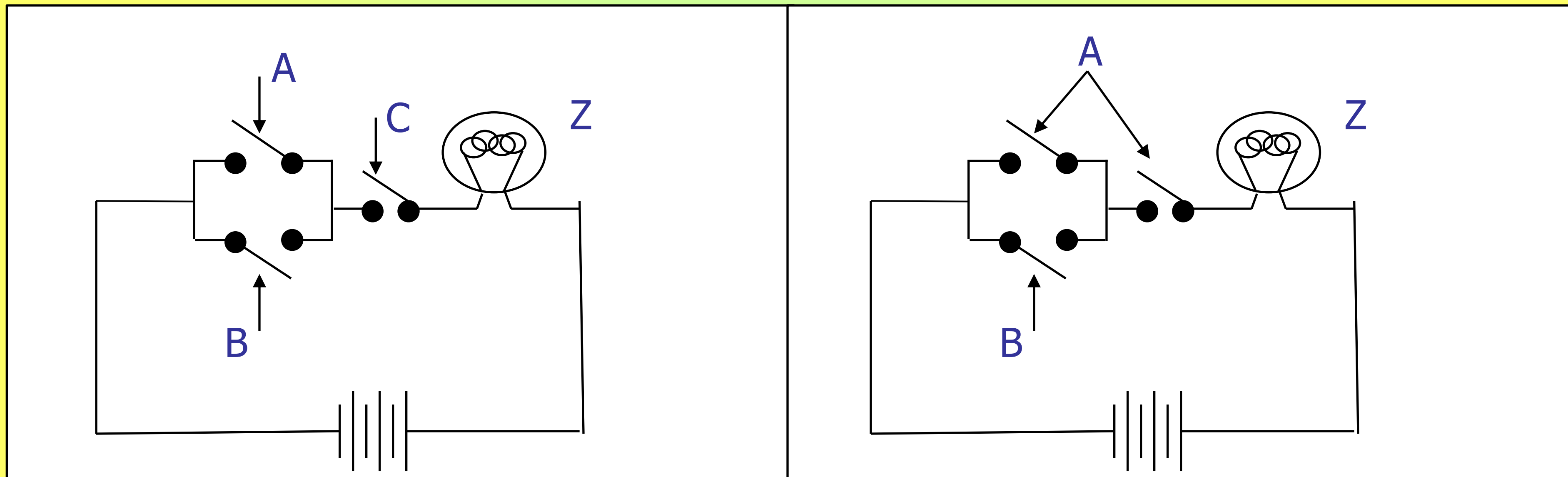
ΕΦΑΡΜΟΓΕΣ:

Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ, ΗΛΕΚΤΡΟΝΙΚΑ ΙΣΧΥΟΣ, ΔΙΑΣΤΗΜΙΚΕΣ ΕΦΑΡΜΟΓΕΣ, ΜΕΤΑΦΟΡΕΣ, ΣΥΣΚΕΥΕΣ ΚΑΘΗΜΕΡΙΝΗΣ ΧΡΗΣΗΣ κλπ.

Περισσότερα για τα τρανζίστορ στο μάθημα της Ηλεκτρονικής στο επόμενο εξάμηνο……..

ΕΠΙΠΕΔΑ ΜΕΛΕΤΗΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Σύνθεση συναρτήσεων



Όσο πιο πολύπλοκο γίνεται το σχέδιο, τόσο πιο προφανές γίνεται ότι χρειαζόμαστε κάποιο τυπικό τρόπο, για να βρούμε πότε το Ζ θα γίνει 1.

Χρειαζόμαστε συνεπώς μια άλγεβρα !

Τι είναι μια άλγεβρα ?

- **Μια μαθηματική δομή :**
  - **Σύνολο ψηφίων & αναπαραστάσεων**
  - **Σύνολο τελεστών & προτεραιότητες**
  - **Αξιώματα**
  - **Θεωρήματα**
  - **Συναρτήσεις**

- **Αλγεβρα των φυσικών αριθμών**
  - **Ψηφίο $\in$ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, Αναπαραστάσεις = ∞**
  - **Τελεστές : +, -, *, /   Προτεραιότητες : {,[,(, /, *, -, +**
  - **Αξιώματα : ουδετερότητα του 0 στη +, του 1 στον *, προσεταιριστικότητα του +, επιμεριστικότητα του * ως προς το +, ...**
  - **Θεωρήματα : x+x = 2x**
  - **Συναρτήσεις f(x,y) = 3x+5y**

Αλγεβρα Boole

- Το 1854 (!!!) ο Αγγλος μαθηματικός George Boole εισήγαγε μια άλγεβρα δύο τιμών : αλήθεια και ψέμα.
- Προτάθηκε για το λογισμό της αλήθειας ή του ψέμματος προτάσεων
- Σχεδόν ένα αιώνα αργότερα (1938), ο Claude Shannon διαπιστώνει ότι απλά αντιστοιχίζοντας τις "αλήθεια" και "ψέμμα" στο "κλειστός διακόπτης" και "ανοικτός διακόπτης", μπορεί να εφαρμόσει τα όσα ανέπτυξε ο Boole και σε κυκλώματα διακοπτών.
- Ετσι η άλγεβρα Boole έγινε "switching algebra".
- Εμείς θα πάμε απλά ένα βήμα πιο πέρα. Αντί για "κλειστός διακόπτης" και "ανοικτός διακόπτης" θα χρησιμοποιούμε τις τιμές δυαδικών μεταβλητών "1" και "0" αντίστοιχα.
- Ετσι έχουμε τη δυαδική άλγεβρα (λογική άλγεβρα).

Αξιώματα

- Δίτιμη άλγεβρα. Κάθε στοιχείο της X $\in$ {0,1}
- Υπαρξη συμπληρώματος (α')
  - Αν α = 0 => α' = 1,   Αν α = 1 => α' = 0
- Δύο λογικές πράξεις :
  - Λογική σύζευξη (and / και) : • (το σύμβολο μπορεί να παραλείπεται σε απλή παράταξη μεταβλητών)
  - Λογική διάζευξη (or / ή) : +
- Οι λογικές πράξεις ακολουθούν τους εξής κανόνες :
  - 0 • 0 = 0,   1 + 1 = 1
  - 1 • 1 = 1,   0 + 0 = 0
  - 0 • 1 = 1 • 0 = 0,   1 + 0 = 0 + 1 = 1
- Σε μία σύνθετη συνάρτηση των δύο λογικών πράξεων, οι προτεραιότητες είναι : {, [, (, ', •, +

Θεωρήματα μιας μεταβλητής

(Απόδειξη με εξέταση όλων των δυνατών τιμών της μεταβλητής)

- $X + 0 = X$, $X \bullet 1 = X$ (Ουδέτερα στοιχεία)
- $X + 1 = 1$, $X \bullet 0 = 0$ (Απορροφητικά στοιχεία)
- $X + X = X$, $X \bullet X = X$ (Αυτοαπορρόφησης)
- $(X')' = X$ (Διπλοαντιστροφής)
- $X + X' = 1$ $X \bullet X' = 0$ (Συμπληρωματικά στοιχεία)

Θεωρήματα δύο και τριών μεταβλητών (1/2)

(Απόδειξη με εξέταση όλων των δυνατών τιμών των δύο μερών)

- **X + Y = Y + X, X • Y = Y • X (Αντιμεταθετική)**
- **(X + Y) + Z = X + (Y + Z), (X • Y) • Z = X • (Y • Z) (Προσεταιριστική)**
  - **Συμπέρασμα :** ***Σε λογικά γινόμενα ή λογικά αθροίσματα, η χρήση παρενθέσεων είναι προαιρετική. Για παράδειγμα μπορώ να γράφω w + x + y + z αφού όπως κι αν υπολογιστεί αυτή η έκφραση θα πάρω το ίδιο λογικό αποτέλεσμα.***
- **X • Y + X • Z = X • (Y + Z), (X + Y) • (X + Z) = X + (Y • Z) (Επιμεριστική)**
  - **Προσοχή :**
    1. **Η έκφραση X • Y + X • Z είναι ισοδύναμη με την (X • Y) + (X • Z). Θυμηθείτε τη προτεραιότητα τελεστών.**
    2. **Ισχύουν και οι 2 επιμερισμοί. Στην άλγεβρα των αριθμών ισχύει μόνο του x ως προς το +.**

Θεωρήματα δύο και τριών μεταβλητών (2/2)

(Απόδειξη με εξέταση όλων των δυνατών τιμών των δύο μερών)

- **X + X • Y = X,** **X • (X + Y) = X** **(Κάλυψης)**
- **X • Y + X • Y'= X** **(X + Y) • (X + Y') = X** **(Συνδυασμών)**
  - ***Συμπέρασμα : Σε ένα άθροισμα γινομένων, που κάθε γινόμενο έχει κ μεταβλητές, αν υπάρχουν όλοι οι συνδυασμοί τιμών των κ-1 μεταβλητών τότε μπορούν να διαγραφούν από την έκφραση.***
- **X • Y + X' • Z + Y • Z = X • Y + X' • Z,**
- **(X +Y) • (X' + Z) • (Y + Z) = (X +Y) • (X' + Z)** **(Συναίνεσης)**

Θεωρήματα γενικευμένου αριθμού μεταβλητών

- $X + X + \ldots + X = X$, $X \bullet X \bullet \ldots \bullet X = X$ (Αυτοαπορρόφησης)

- $(X_1 + X_2 + \ldots + X_n)' = X_1' \bullet X_2' \bullet \ldots \bullet X_n'$
- $(X_1 \bullet X_2 \bullet \ldots \bullet X_n)' = X'_1 + X'_2 + \ldots + X'_n$ (Θεωρήματα De Morgan)

- $[F(X_1, X_2, \ldots, X_n, +, \bullet)]' = F(X'_1, X'_2, \ldots, X'_n, \bullet, +)$ (Γενικευμένο θεώρημα De Morgan)
  - Παράδειγμα : Δίδεται η $F = (W' \bullet X) + (X \bullet Y) + [W \bullet (X' + Z')]$.
  - Τότε $F' = ((W')' + X') \bullet (X' + Y') \bullet [W' + (X \bullet Z)] =$

    $= (W + X') \bullet (X \bullet Y)' \bullet [W' + (X \bullet Z)]$

- $F(X_1, X_2, \ldots, X_n) = X_1 \bullet F(1, X_2, \ldots, X_n) + X_1' \bullet F(0, X_2, \ldots, X_n)$
- $F(X_1, X_2, \ldots, X_n) = [X_1 + F(0, X_2, \ldots, X_n) \bullet [X_1' + F(1, X_2, \ldots, X_n)]$ (Θεωρήματα Shannon)
  - Παράδειγμα : Δίδεται η $F(X, W, Z) = X + W \bullet Z$.
  - Τότε $F(0, W, Z) = W \bullet Z$ και $F(1, W, Z) = 1$.
  - Αρα $F = X \bullet 1 + X' \bullet W \bullet Z$ καθώς και $F = (X + W \bullet Z) \bullet (X' + 1)$

Αναπαράσταση μεγεθών
Κωδικοποίηση

Συστήματα αριθμών

- **Δεκαδικό σύστημα**

$$D_{10} = \left(d_n 10^n + d_{n-1} 10^{n-1} + \ldots d_1 10^1 + d_0 10^0 + d_{-1} 10^{-1} + \ldots d_{-n} 10^{-n}\right)$$

Παράδειγμα

$$503{,}14 =$$

$$500 + 0 + 3 + \frac{1}{10} + \frac{4}{100} =$$

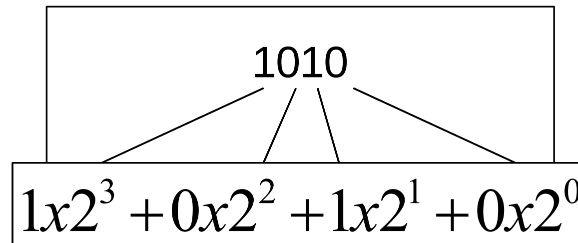
$$5 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2}$$

Δυαδικό σύστημα

Στο δυαδικό σύστημα, που έχει βάση το 2, υπάρχουν δύο ψηφία, το 0 και το 1:

$$B_2 = b_n 2^n + b_{n-1} 2^{n-1} + \ldots b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + \ldots b_{-m} 2^{-m}$$

Παράδειγμα



Γενικά ένας δυαδικός αριθμός με ***n*** ψηφία μπορεί να παραστήσει ένα εύρος από ***2ⁿ*** δεκαδικούς αριθμούς

2 ψηφία (0_3), 5 ψηφία (0_31), 8 ψηφία (0_255)

Μετατροπή δεκαδικού σε δυαδικό

- Μετατροπή ενός ακέραιου δεκαδικού σε δυαδικό: **χρησιμοποιείται η διαδικασία της διαδοχικής διαίρεσης**

- *Παράδειγμα:*

Μετατροπή του 19_{10} στον αντίστοιχο δυαδικό

19/2= πηλίκο 9 και υπόλοιπο 1 άρα b_0=1
9/2= πηλίκο 4 και υπόλοιπο 1 άρα b_1=1
4/2= πηλίκο 2 και υπόλοιπο 0 άρα b_2=0
2/2= πηλίκο 1 και υπόλοιπο 0 άρα b_3=0
1/2= πηλίκο 0 και υπόλοιπο 1 άρα b_4=1

B_2=10011=19_{10}

- Μετατροπή του κλασματικού μέρους ενός δεκαδικού αριθμού στον αντίστοιχο δυαδικό:

χρησιμοποιείται η διαδικασία των διαδοχικών πολλαπλασιασμών. Επαναλαμβάνεται η διαδικασία μέχρι να προκύψει κλασματικό μέρος μηδέν ή να επιτευχθεί η επιθυμητή ακρίβεια.

Παράδειγμα:

Μετατροπή του 0,375 στον αντίστοιχο δυαδικό

0,375 x 2 = 0,75, ακέραιο μέρος 0, κλασματικό 0,75 $b_{-1}=0$

0,75 x 2 = 1,5, ακέραιο μέρος 1, κλασματικό 0,5 $b_{-2}=1$

0,5 x 2 = 1,0, ακέραιο μέρος 1, κλασματικό 0 $b_{-2}=1$

$B_2= : ,011_2$

- *Μετατροπή του 28,375 στον αντίστοιχο δυαδικό*
- *Απάντηση:* $B_2= : 11100,011_2$

Βασικές λογικές πράξεις – λογικές πύλες

- Μία λογική πράξη μεταξύ μεταβλητών είναι μία συνάρτηση που ορίζεται από έναν πίνακα αληθείας (truth table). Το ηλεκτρικό κύκλωμα που εκτελεί μία λογική πράξη ονομάζεται ***λογική ή ψηφιακή πύλη*** και παριστάνεται από ένα σύμβολο. Τα δυαδικά ψηφία 1 και 0, που ουσιαστικά παριστάνουν τις δύο καταστάσεις αληθής (true), ψευδής (false), στη φυσική τους υπόσταση είναι δυο διακριτά επίπεδα ηλεκτρικής τάσης (συνήθως στην ιδανική περίπτωση 5V και 0V).

Δυνατοί πίνακες αληθείας στο δυαδικό σύστημα

- Ένας πίνακας αληθείας παριστάνει τη συνάρτηση μεταξύ των εισόδων και της εξόδου ενός λογικού συστήματος. Για δυο εισόδους υπάρχουν τέσσερις πιθανοί συνδυασμοί πραγματικών τιμών: ***FF, FT, TF, TT***
- Επειδή κάθε δυνατή είσοδος μπορεί να δώσει δύο διαφορετικές εξόδους ***(F, T)*** συνεπάγεται ότι οι δυνατοί πίνακες αληθείας για ένα λογικό σύστημα δύο εισόδων είναι:

$$2^4 = 16$$

Όλοι οι πίνακες αληθείας για δύο εισόδους Α. Β και μία έξοδο Ζ

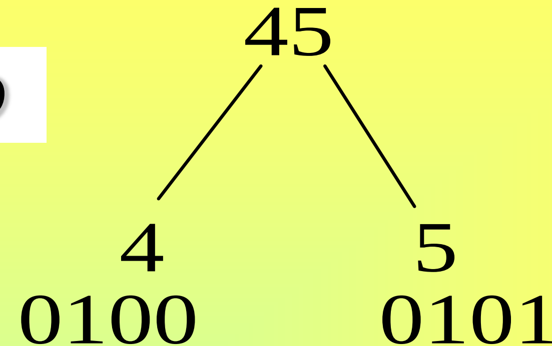
	τιμές εισόδου					
A	F	F	T	T		
B	F	T	F	T	Συνάρτηση (έξοδος Ζ)	Σύμβολο
0	F	F	F	F	πάντοτε 0	0
1	F	F	F	T	AND	$A \cdot B$
2	F	F	T	F	-	-
3	F	F	T	T	είσοδος Α	A
4	F	T	F	F	-	-
5	F	T	F	T	είσοδος Β	B
6	F	T	T	F	XOR	$A \oplus B$
7	F	T	T	T	OR	$A + B$
8	T	F	F	F	NOR	$\overline{A + B}$
9	T	F	F	T	XNOR	$\overline{A \oplus B}$
10	T	F	T	F	Not B	$\overline{B}$
11	T	F	T	T	-	-
12	T	T	F	F	Not A	$\overline{A}$
13	T	T	F	T	-	-
14	T	T	T	F	NAND	$\overline{A \cdot B}$
15	T	T	T	T	πάντοτε 1	1

Άλλοι τρόποι δυαδικής κωδικοποίησης

- Εκτός από την κανονική δυαδική κωδικοποίηση υπάρχουν και άλλοι τρόποι δυαδικής κωδικοποίησης οι οποίοι χρησιμοποιούνται σε διάφορες περιπτώσεις:
- **Κωδικοποίηση BCD (Binary Coded Decimal)**

Η κωδικοποίηση καθιστά δυνατή την απλή μετατροπή μεταξύ δυαδικού και δεκαδικού αριθμού. Κάθε ψηφίο ενός δεκαδικού αριθμού αντικαθίσταται από 4 bits του αντίστοιχου δυαδικού του

Μετατροπή του 45_{10} σε BCD



45_{10}=01000101_{BCD}

Μετατροπή από BCD σε δεκαδικό

Η δυαδική λέξη χωρίζεται σε ομάδες των 4bits ξεκινώντας από το λιγότερο σημαντικό ψηφίο. Κατόπιν η κάθε ομάδα μετατρέπεται στον αντίστοιχο δεκαδικό

Μετατροπή 1010011_{BCD} σε δεκαδικό

Πρόσθεση μηδενικού . Χωρισμός σε ομάδες των 4. Μετατροπή της κάθε ομάδας στον αντίστοιχο δεκαδικό

$[0101][0011]_{BCD}=53_{10}$

- **Κώδικας Gray**

Συχνά χρησιμοποιείται σε ηλεκτρονικά κυκλώματα για την αποφυγή προβλημάτων που θα μπορούσαν να προκύψουν εάν χρησιμοποιούνταν η απευθείας δυαδική κωδικοποίηση. Για παράδειγμα, σε μετρήσεις της θέσης ενός αντικειμένου, θα μπορούσε να φαίνεται ότι γειτονικές θέσεις του αντικειμένου διαφέρουν περισσότερο από ένα bit, εάν χρησιμοποιηθεί η απευθείας δυαδική κωδικοποίηση.

Decimal	Gray Code	
0	0	
1	1	
2	11	Set bit 1. Reflect bit 0
3	10	Set bit 2. Reflect bits 1 and 0
4	110	
5	111	
6	101	
7	100	
8	1100	Set bit 3. Reflect bits 2, 1 and 0
9	1101	
10	1111	
11	1110	
12	1010	
13	1011	
14	1001	
15	1000	
16	11000	Set bit 4. Reflect bits 3, 2, 1 and 0

Imagine a mirror is placed under the reflected bits.

Για νε μετατρέψουμε αριθμούς σε κώδικα Gray κάνουμε χρήση της συνάρτησης XOR, η οποία έχει πίνακα αλήθειας:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Και χρησιμοποιούμε τον παρακάτω αλγόριθμο.

$$g_n = b_n$$

$$g_{n-1} = b_n \oplus b_{n-1}$$

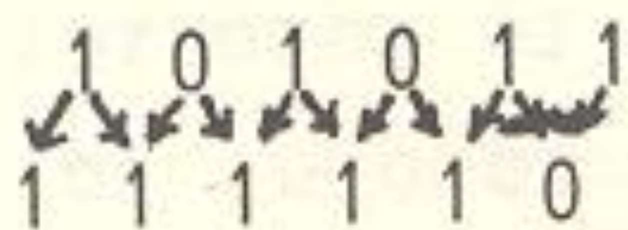
$$g_{n-2} = b_{n-1} \oplus b_{n-2}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots$$

$$g_1 = b_2 \oplus b_1$$

<u>Παράδειγμα</u> : Να μετατραπεί ο 43_{10} σε κώδικα GRAY.

$$43_{10} = 101011_2$$



δηλαδή $43_{10} = 101011_2 = 111110_{GRAY}$.

$$b_n = g_n$$

$$b_{n-1} = b_n \oplus g_{n-1}$$

$$b_{n-2} = b_{n-1} \oplus g_{n-2}$$

. . . .

$$b_1 = b_2 \oplus g_1$$

ΚΩΔΙΚΑΣ GRAY (5 bits)

0	0	0	0	0	0		1	1	0	0	0	16
0	0	0	0	1	1		1	1	0	0	1	17
0	0	0	1	1	2		1	1	0	1	1	18
0	0	0	1	0	3		1	1	0	1	0	19
0	0	1	1	0	4		1	1	1	1	0	20
0	0	1	1	1	5		1	1	1	1	1	21
0	0	1	0	1	6		1	1	1	0	1	22
0	0	1	0	0	7		1	1	1	0	0	23
0	1	1	0	0	8		1	0	1	0	0	24
0	1	1	0	1	9		1	0	1	0	1	25
0	1	1	1	1	10		1	0	1	1	1	26
0	1	1	1	0	11		1	0	1	1	0	27
0	1	0	1	0	12		1	0	0	1	0	28
0	1	0	1	1	13		1	0	0	1	1	29
0	1	0	0	1	14		1	0	0	0	1	30
0	1	0	0	0	15		1	0	0	0	0	31
						1	1	0	0	0	0	32

- **Κώδικες με ανίχνευση σφάλματος**

  Στα ψηφιακά συστήματα, υπάρχουν περιπτώσεις όπου κατά την παραγωγή δεδομένων και την επεξεργασία αυτών, εμφανίζονται σφάλματα. Για παράδειγμα κάποιο ψηφίο 1, ενός συνόλου δυαδικών ψηφίων, μπορεί να μετατραπεί σε ψηφίο 0, είτε κατά το στάδιο της μετάδοσης, είτε γιατί το ψηφιακό σύστημα δεν λειτούργησε σωστά. Μία απλή μέθοδος, ανίχνευσης του σφάλματος, είναι η χρήση του κώδικα ανίχνευσης λάθους, η οποία χρησιμοποιεί ένα επιπλέον ψηφίο ισοτιμίας (parity bit).

- ***Κώδικες ισοτιμίας***

- *Δυο είδη* {
  - ***περιττή ισοτιμία***
  - ***άρτια ισοτιμία***

- *Κώδικας περιττής ισοτιμίας*

  Το ψηφίο ισοτιμίας είναι 0 αν το σύνολο των ψηφίων, 1, είναι περιττό. Το ψηφίο ισοτιμίας είναι 1 αν το σύνολο των ψηφίων, 1, είναι άρτιο.

  Για παράδειγμα η δυαδική λέξη **010001** έχει αριθμό ψηφίων ‘1’ άρτιο, συνεπώς θα μεταδοθεί με ψηφίο ισοτιμίας ‘1’, είτε: **1 | 010001**

- *Κώδικας άρτιας ισοτιμίας*

  Αντίστροφος της περιττής ισοτιμίας. Το ψηφίο ισοτιμίας είναι 1 αν το σύνολο των ‘1’ είναι περιττό. Το ψηφίο ισοτιμίας είναι 0 αν το σύνολο των ‘1’ είναι άρτιο

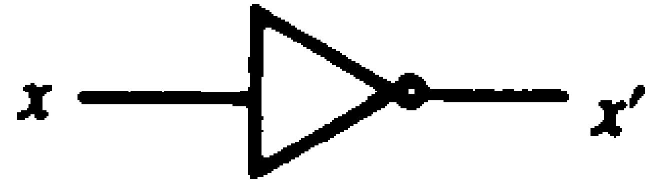
  Για παράδειγμα η δυαδική λέξη 10110 έχει αριθμό ψηφίων ‘1’ περιττό, συνεπώς θα μεταδοθεί με ψηφίο ισοτιμίας ‘1’, είτε: **1 | 10110**

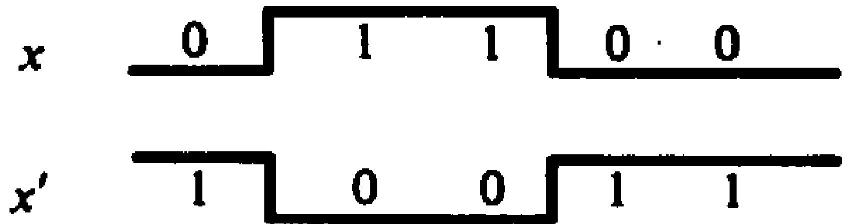
Γιατί τόση θεωρία ?

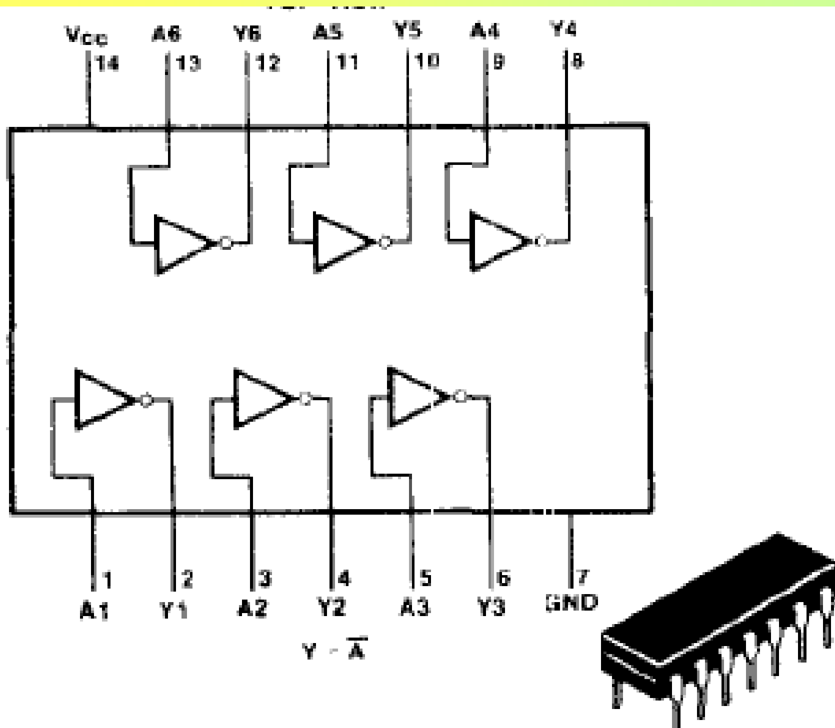
- **Όσα είδαμε έχουν άμεση εφαρμογή στο πραγματικό κόσμο.**
- **Υπάρχουν έτοιμοι ψηφιακοί σχεδιασμοί οι οποίοι υλοποιούν τις βασικές λογικές συναρτήσεις**
- **Οι σχεδιασμοί αυτοί ονομάζονται *ψηφιακές πύλες*.**
- **Θα εισάγουμε :**
  - **Κάποια σχηματικά για την απεικόνιση αυτών.**
  - **Ενα πίνακα που δείχνει τη λογική συνάρτηση που επιτελείται από το συγκεκριμένο ψηφιακό κύκλωμα. Ο πίνακας αυτός είναι ο *πίνακας αληθείας* της συνάρτησης, άρα και του ψηφιακού σχεδιασμού.**

Ο αντιστροφέας

- Παράγει το συμπλήρωμα μιας δυαδικής μεταβλητής.
- Μπορούμε αντί για το πλήρες σχηματικό του αντιστροφέα, να χρησιμοποιούμε μόνο το κύκλο.
- Είναι διαθέσιμο ως ψηφιακό κύκλωμα σε 6άδες, εντός ενός ολοκληρωμένου με κωδικό 7404.



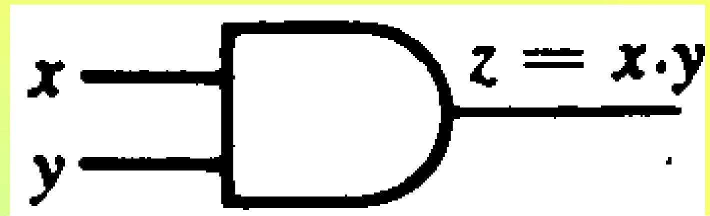


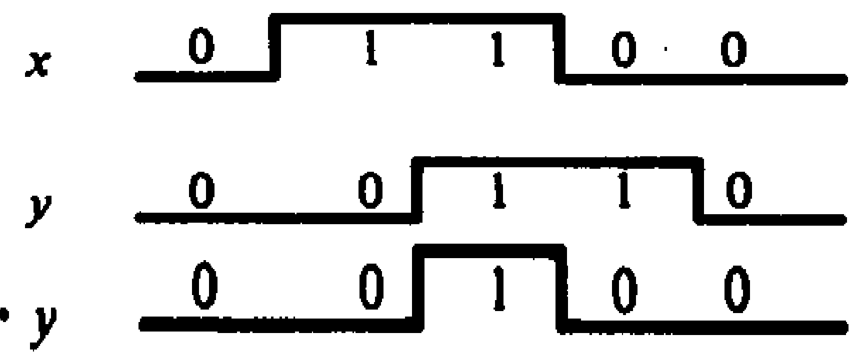


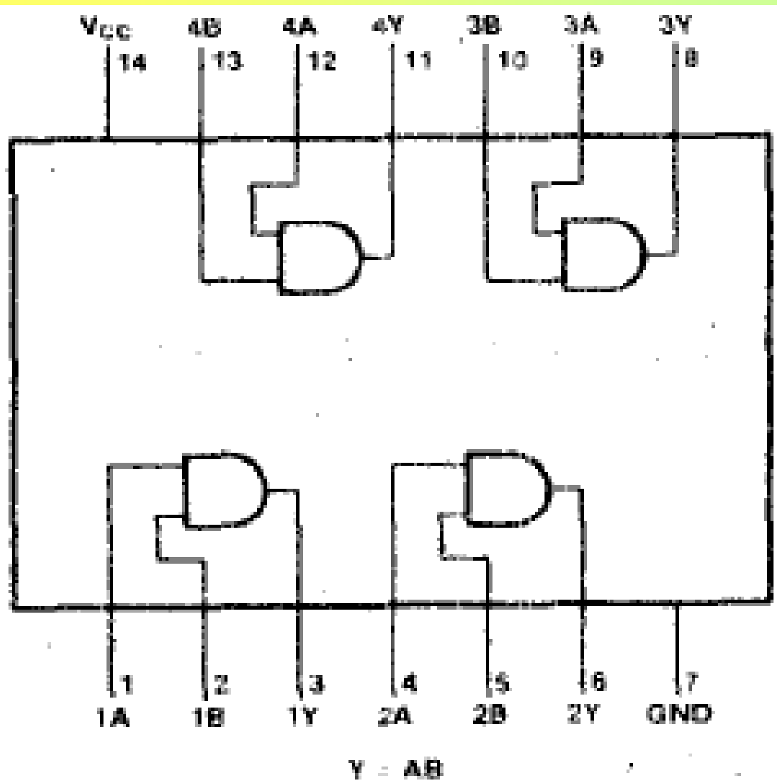
x	x'
0	1
1	0

Η πύλη AND δύο μεταβλητών

- Εκτελεί το λογικό AND των δύο μεταβλητών.
- Είναι διαθέσιμο ως ψηφιακό κύκλωμα σε 4άδες, εντός ενός ολοκληρωμένου με κωδικό 7408.



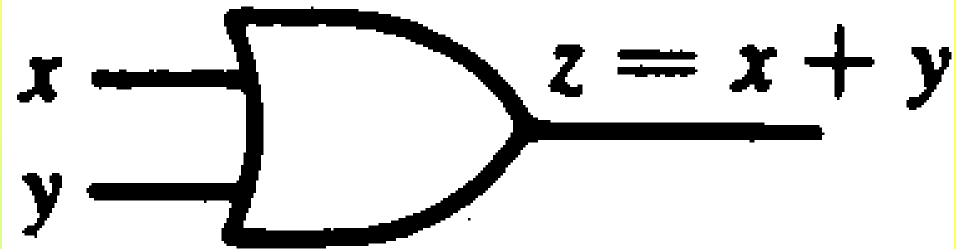


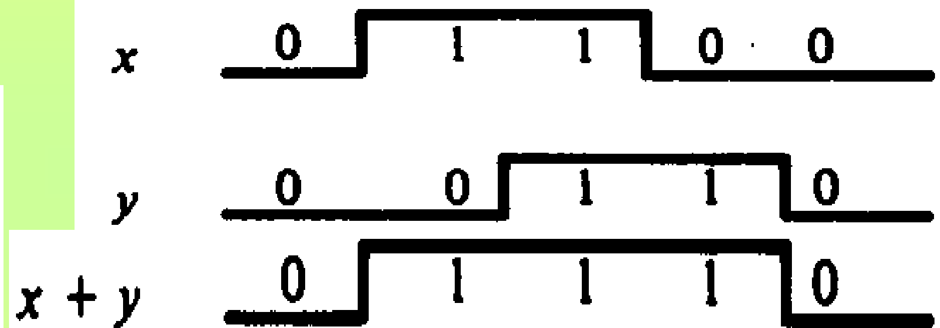


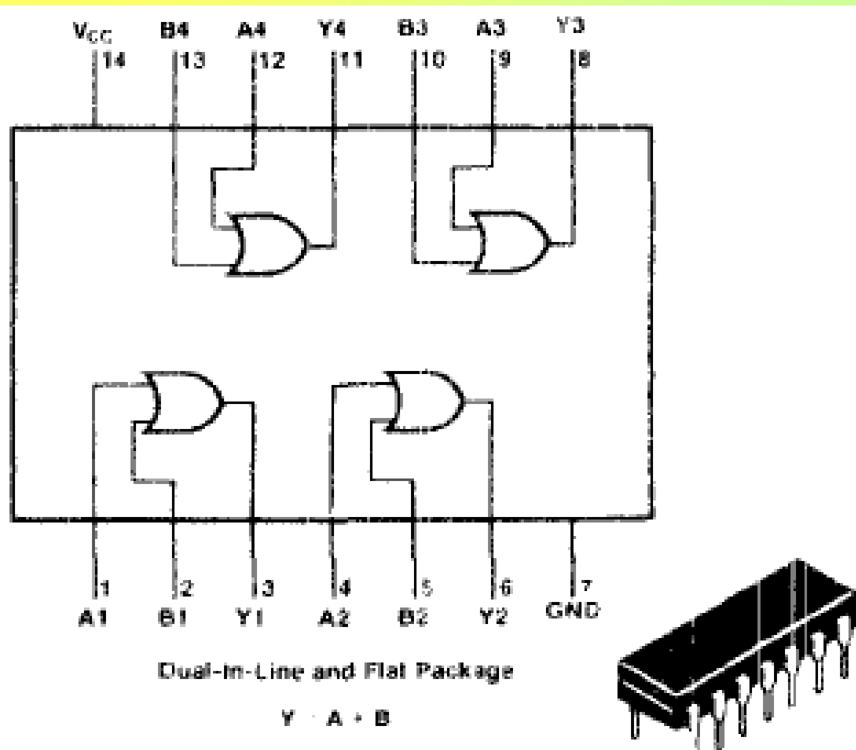
x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Η πύλη OR δύο μεταβλητών

- Εκτελεί το λογικό OR των δύο εισόδων.
- Είναι διαθέσιμο ως ψηφιακό κύκλωμα σε 4άδες, εντός ενός ολοκληρωμένου με κωδικό 7432.



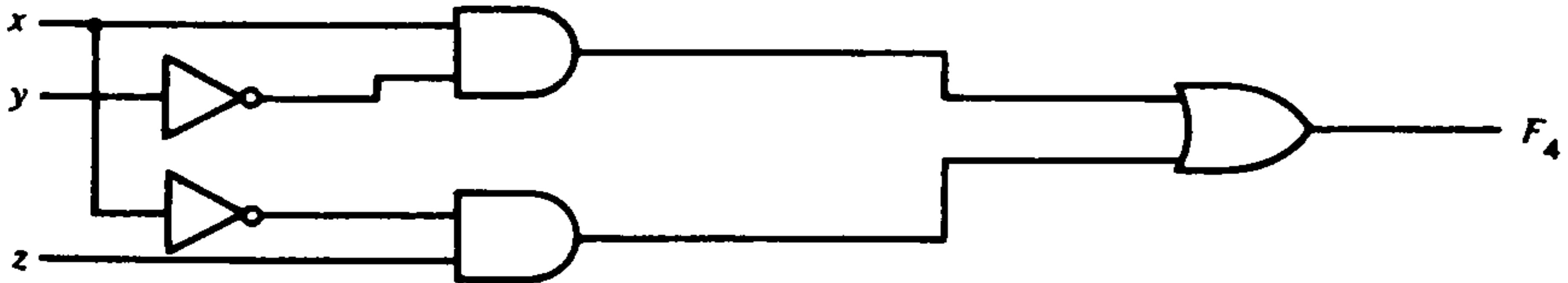




x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Λογικό διάγραμμα

- Η χρήση των σχηματικών των διαφόρων πυλών και η διασύνδεσή τους οδηγεί σε ένα **λογικό διάγραμμα** που περιγράφει κάποια πιο σύνθετη συνάρτηση.
- Για παράδειγμα :



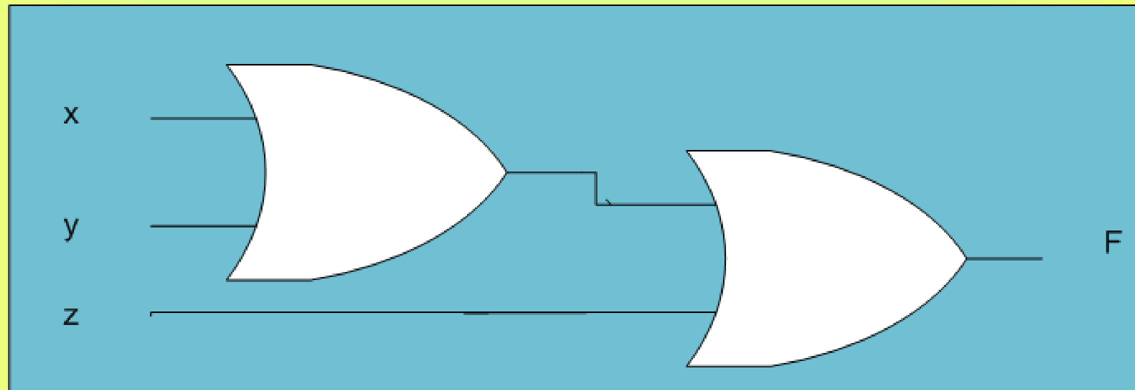
- Το παραπάνω είναι ένα διάγραμμα που περιγράφει τη συνάρτηση :

$$F_4(x, y, z) = x' \bullet z + y' \bullet x$$

- Ποιο είναι το λογικό διάγραμμα της $G(X, W, Y, Z) = [X' \bullet Y + X] \bullet (Z + W')$;

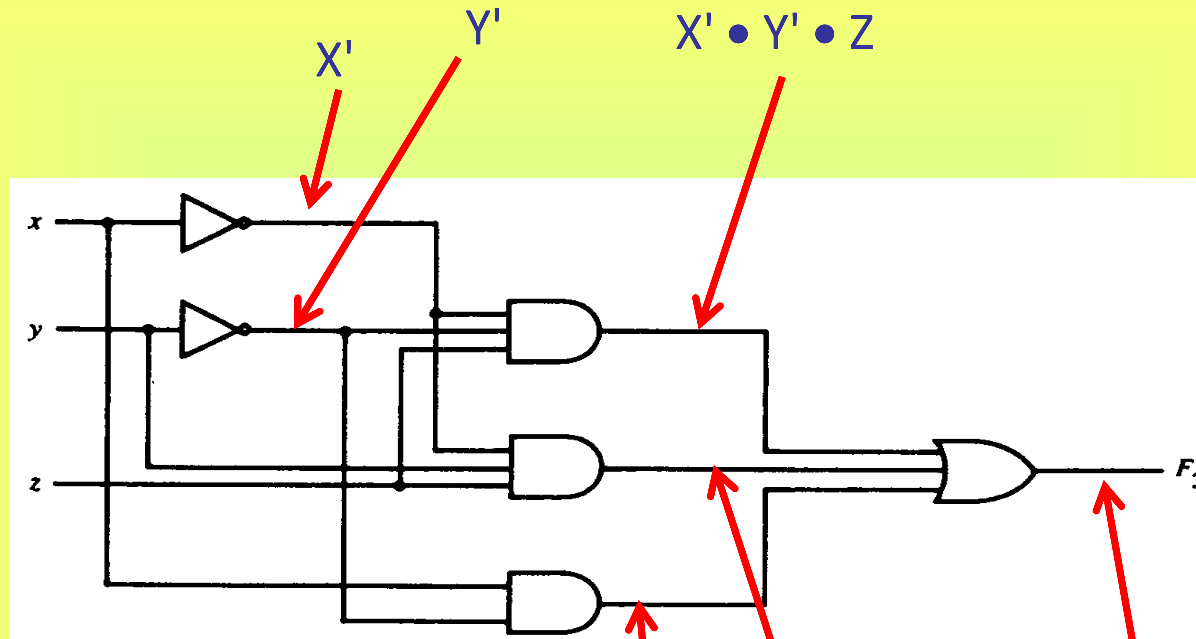
Επέκταση σε περισσότερες εισόδους

- Αν ήθελα το λογικό διάγραμμα της F(x, y, z) = x + y + z πως θα το έφτιαχνα ?
- Από τη θεωρία μου γνωρίζω ότι F(x, y, z) = x + y + z = (x + y) + z και συνεπώς θα μπορούσα να χρησιμοποιήσω κάτι τέτοιο :



- Μήπως υπάρχουν έτοιμες και πύλες περισσότερων εισόδων ?
- Φυσικά ! Αυτό όμως δε σημαίνει ότι υπάρχουν οσωνδήποτε εισόδων.
- Κι εδώ πρέπει να διαχωρίσουμε το πραγματικό από τον ιδεατό κόσμο :
    - Στον ιδεατό κόσμο μπορούμε να φτιάχνουμε λογικά διαγράμματα με πύλες όσων εισόδων θέλουμε.
    - Στη πράξη αυτά θα υλοποιηθούν με όσα πραγματικά υπάρχουν.

Ποια συνάρτηση επιτελεί αυτό το κύκλωμα ?



X • Y'

X' • Y • Z

X • Y' + X' • Y • Z + X' • Y' • Z

Ποιες άλλες συναρτήσεις και πύλες υπάρχουν ?

- Πόσες διαφορετικές συναρτήσεις των ν μεταβλητών υπάρχουν ?
  - Μια συνάρτηση ν μεταβλητών, έχει 2^{ν} πιθανές τιμές εισόδου.
  - Διαφορετική συνάρτηση ⇔ διαφορετική έξοδος έστω και για κάποιον συνδυασμό εισόδου. Αφού η έξοδός μου για κάθε συνδυασμό εισόδου μπορεί να είναι 0 ή 1 θα υπάρχουν $2^{2^{\nu}}$ συναρτήσεις.
- Για ν =2 ποιες είναι οι συναρτήσεις που υπάρχουν και ποιες από αυτές είναι χρήσιμες ?

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Σύμβολο τελεστή			·	/		/		⊕	+	↓	⊙	′	⊂	′	⊃	↑	

1. Δυο σταθερές 0, 1.
2. Τέσσερις unary συμπληρώματος/μεταφοράς.
3. Δέκα συναρτήσεις με δυαδικούς τελεστές.

Μας ενδιαφέρουν επίσης οι :

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Σύμβολο τελεστή			$\cdot$	/		/		$\oplus$	$+$	$\downarrow$	$\odot$	$'$	$\subset$	$'$	$\supset$	$\uparrow$	

- F_{14} που είναι συμπληρωματική της AND. Θα τη λέμε NAND (not – AND).
- F_8 που είναι συμπληρωματική της OR. Θα τη λέμε NOR (not – OR).
- F_6 που μας δίνει 1 μόνο όταν μόνο 1 είσοδος είναι στο λογικό 1 (για περισσότερες εισόδους, όταν ο αριθμός των 1 στις εισόδους είναι περιττός). Θα την ονομάζουμε αποκλειστικό-OR (eXclusive-OR) – XOR.
- F_9 που μας δίνει 1 μόνο όταν μόνο 0 ή 2 είσοδοι είναι στο λογικό 1 (για περισσότερες εισόδους όταν ο αριθμός των 1 στις εισόδους είναι άρτιος). Θα την ονομάζουμε συνάρτηση ισοδυναμίας (not eXclusive-OR) – XNOR.

Κόστος πυλών

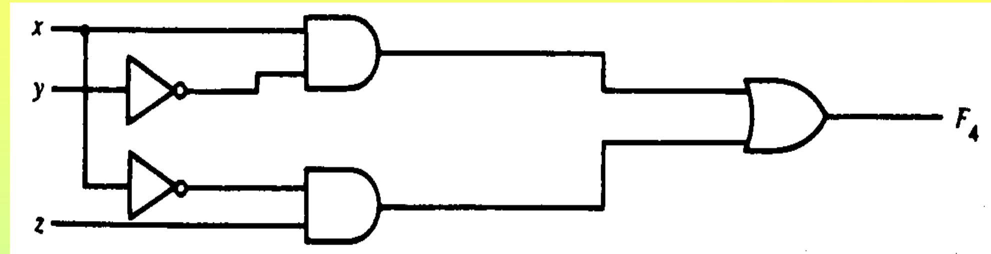
- Κοστίζουν όλες οι πύλες το ίδιο ?
- Όχι. Άλλωστε είναι χαρακτηριστικό ότι σε ένα ολοκληρωμένο κύκλωμα έχουμε 6 inverters και μόνο 4 OR, AND, …
- Το κόστος μιας πύλης εξαρτάται :
  - Από τον αριθμό των εισόδων της. Περισσότεροι είσοδοι => ↑ κόστος.
  - Από τη συνάρτηση που επιτελεί :
    - NAND, NOR οι πιο απλές.
    - AND, OR, ελάχιστα πιο πολύπλοκες (σε επίπεδο transistor).
    - XOR, XNOR αρκετά πιο πολύπλοκες
    - Μεγαλύτερη πολυπλοκότητα => ↑ κόστος

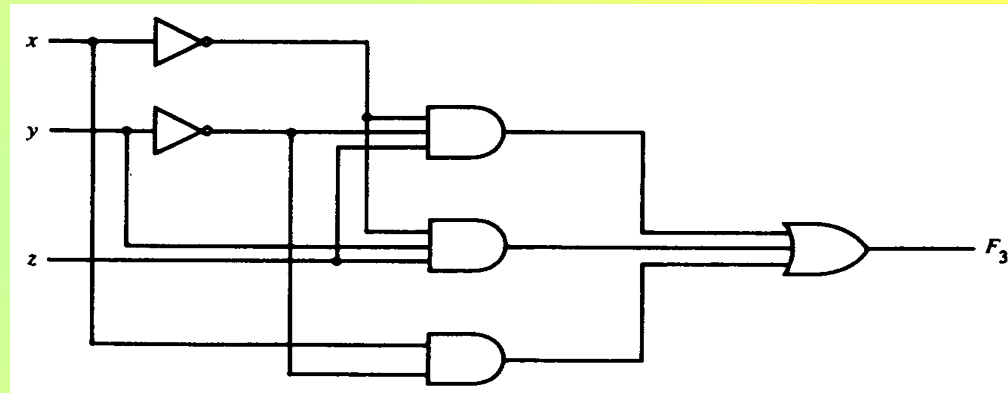
Ισοδύναμες και συμπληρωματικές συναρτήσεις

- Δύο συναρτήσεις των κ μεταβλητών είναι ισοδύναμες, αν για κάθε συνδυασμό τιμών των μεταβλητών εισόδου, οι συναρτήσεις οδηγούν στην ίδια έξοδο.
  - ***Συμπέρασμα 1 : Ισοδύναμες συναρτήσεις μπορεί να έχουν διαφορετικές αλγεβρικές εκφράσεις.***
  - ***Συμπέρασμα 2 : Ισοδύναμες συναρτήσεις μπορεί να έχουν διαφορετικά λογικά διαγράμματα.***
  - ***Συμπέρασμα 3 : Ισοδύναμες συναρτήσεις έχουν τον ίδιο πίνακα αληθείας.***
- Δύο συναρτήσεις των κ μεταβλητών είναι συμπληρωματικές, αν για κάθε συνδυασμό των μεταβήτών εισόδου, οι συναρτήσεις παράγουν συμπληρωματικές τιμές.

Μεταξύ ισοδύναμων συναρτήσεων κάποιες είναι σαφώς προτιμητέες !

- Η F_3 είναι σαφώς πιο πολύπλοκη συνάρτηση από την F_4.
- Χρησιμοποιεί περισσότερες πύλες, μα ταυτόχρονα και πύλες με περισσότερες εισόδους.
- Παράλληλα χρησιμοποιεί 4 διαφορετικά ολοκληρωμένα έναντι 3 της F_4.
- Το μεγάλο συνεπώς ερώτημα που προκύπτει είναι **πως θα βρω τη συνάρτηση με το ελάχιστο κόστος ανάμεσα στις ισοδύναμες ?**
- **Αυτό είναι το πιο ενδιαφέρον σημείο του μαθήματος :** ***απλοποίηση συναρτήσεων***





x	y	z	F_3	F_4
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Η θεωρία παρέχει τη μέθοδο της αλγεβρικής απλοποίησης

- Θυμηθείτε ότι για την F_3 είχαμε ότι : F_3 (X, Y, Z) = X • Y' + X' • Y • Z + X' • Y' • Z
- Από τη θεωρία βάσει του θεωρήματος των συνδυασμών γνωρίζω ότι :
  - X' • Y • Z + X' • Y' • Z = X' • Z
  - Αρα F_3 = X • Y' + X' • Z = F_4.
- Η αλγεβρική απλοποίηση δεν είναι πάντα τόσο εύκολη !
- Ακόμα χειρότερα, ποτέ δε γνωρίζω αν έχοντας κάνει κάποια στάδια απλοποίησης έχω βρει την απολύτως ελάχιστη ισοδύναμη συνάρτηση.
- Η αλγεβρική απλοποίηση πρέπει να χρησιμοποιείται για συναρτήσεις πολύ λίγων μεταβλητών από έμπειρους σχεδιαστές.

Παραδείγματα αλγεβρικής απλοποίησης

- $F(X, Y, Z) = X \bullet Y' \bullet Z + X' \bullet Y \bullet Z + Y \bullet Z =$

  $= X \bullet Y' \bullet Z + Y \bullet Z$ (Απορρόφηση)

  $= Z \bullet (X \bullet Y' + Y)$ (Επιμεριστική)

  $= Z \bullet (X + Y)$ (1^o αποδειχθέν θεώρημα)

- Υλοποιείστε με τον ελάχιστο αριθμό πυλών τη $F(X, Y, Z) = X \bullet Y' \bullet Z + X \bullet Y' \bullet Z + X \bullet Y \bullet Z'$
  - $F(X, Y, Z) = X \bullet Y' \bullet Z + X \bullet Y' \bullet Z + X \bullet Y \bullet Z' =$

    $= X \bullet Y' \bullet Z + X \bullet Y \bullet Z'$ (Αυτοαπορρόφηση)

    $= X \bullet (Y' \bullet Z + Y \bullet Z')$ (Επιμεριστική)

    $= X \bullet (Y \oplus Z)$ (συνάρτηση XOR)

Προβλήματα & Αλγόριθμοι

- Η αλγεβρική απλοποίηση
  - Είναι δύσκολη
  - Μη ντετερμινιστική
  - Χωρίς σίγουρο αποτέλεσμα
- Θα ήθελα συνεπώς μια ντετερμινιστική μεθοδολογία.
- Στη γλώσσα των υπολογιστών μια μεθοδολογία που αποτελείται από μικρά κατανοητά βήματα και η οποία αν ακολουθηθεί παράγει τη λύση σε κάποιο πρόβλημα ονομάζεται ***αλγόριθμος***.
- Για να εφαρμοστεί κάποιος αλγόριθμος όμως απαιτείται να υπάρχει μια σταθερή αρχική μορφή του προβλήματος.
- Πριν λοιπόν δώσουμε αλγόριθμο απλοποίησης, πρέπει πρώτα να εισάγουμε πρότυπες μορφές έκφρασης μιας συνάρτησης.
- Σε αυτό θα μας βοηθήσουν οι ελαχιστόροι και οι μεγιστόροι.

Πίνακας αλήθειας

- Ισοδύναμες συναρτήσεις έχουν ποικίλες αλγεβρικές αναπαραστάσεις, ποικίλα λογικά διαγράμματα, *αλλά κοινό πίνακα αλήθειας.*
- Ο πίνακας αλήθειας μας δίνει τη τιμή μιας συνάρτησης για κάθε πιθανό συνδυασμό εισόδων.
- Συνήθως διατάσσουμε τους συνδυασμούς εισόδων σαν αύξοντες δυαδικούς αριθμούς.
- Ο πίνακας αλήθειας μιας λογικής συνάρτησης ν μεταβλητών έχει 2^{ν} γραμμές.

Οροι, αθροίσματα, γινόμενα

- Μια μεταβλητή στη κανονική ή τη συμπληρωματική της μορφή είναι ένας ***όρος***
  - Παραδείγματα όρων : X, X', Y, Z'
- Ενα ***γινόμενο*** είναι είτε ένας όρος είτε το λογικό AND δύο ή περισσότερων όρων
  - Παραδείγματα γινομένων : X, X • Z', Y• X', X • Z' • Y'
- Ενα ***άθροισμα*** είναι είτε ένας όρος είτε το λογικό OR δύο ή περισσοτέρων όρων
  - Παραδείγματα αθροισμάτων : X, X + Z', Y+ X', X + Z' + Y'
  - ***Αθροισμα γινομένων (sum of products – SOP)*** είναι κάθε άθροισμα του οποίου οι όροι είναι γινόμενα
  - Παράδειγμα SOP : X + X • Z' + Y• X' + X • Z' • Y'
  - ***Γινόμενο αθροισμάτων (product of sums – POS)*** είναι κάθε γινόμενο του οποίου οι όροι είναι αθροίσματα.
  - Παράδειγμα POS : X • (X + Z') • (Y+ X') • (X + Z' + Y')
  - ***Κανονικός όρος*** είναι ένα άθροισμα ή ένα γινόμενο, στο οποίο κάθε μεταβλητή (κανονική ή συμπληρωμένη) εμφανίζεται μόνο 1 φορά. Κάθε μη κανονικός όρος μπορεί να μετατραπεί μέσω απλοποίησης σε κανονικό.
  - Κανονικοί όροι : X • Z' • Y', X + Z' + Y'
  - Μη κανονικοί όροι : X • Z' • Y' • X, X + Z' + Y' + Z

Ελαχιστόροι - Μεγιστόροι

- Κάθε κανονικός όρος - γινόμενο μιας συνάρτησης κ μεταβλητών, είναι ***ελαχιστόρος*** αν περιέχει κ όρους. Κάθε συνάρτηση κ μεταβλητών έχει $2^κ$ ελαχιστόρους.
  - Η F(X, Y) έχει τους ελαχιστόρους : X' • Y΄, X΄ • Y, X • Y' και X • Y
  - *Κάθε ελαχιστόρος αντιπροσωπεύει μία γραμμή του πίνακα αληθείας*

- Κάθε κανονικός όρος – άθροισμα μιας συνάρτησης κ μεταβλητών, είναι ***μεγιστόρος*** αν περιέχει κ όρους. Κάθε συνάρτηση κ μεταβλητών έχει $2^κ$ μεγιστόρους.
  - Η F(X, Y) έχει τους μεγιστόρους : X' + Y΄, X΄ + Y, X + Y' και X + Y
  - *Κάθε μεγιστόρος αντιπροσωπεύει μία γραμμή του πίνακα αληθείας*

Ελαχιστόροι – Μεγιστόροι και πίνακας αλήθειας

- Μεταξύ πίνακα αλήθειας και ελαχιστόρων (μεγιστόρων) υπάρχει μια στενή σχέση. Ο ελαχιστόρος (μεγιστόρος) μπορεί να οριστεί σα το γινόμενο (άθροισμα) που μπορεί να γίνει 1 (0) μόνο για τις τιμές εισόδου μιας και μόνο γραμμής του πίνακα αλήθειας.

			Ελαχιστόροι		Μεγιστόροι	
x	y	z	Όρος	Ονομασία	Όρος	Ονομασία
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

- Μπορούμε να ορίσουμε τη δυαδική τιμή που επαληθεύει τον όρο (κάνει 1 τον ελαχιστόρο ή 0 τον μεγιστόρο) σα το διακριτικό του αντίστοιχου όρου.
  - Π.χ. Ελαχιστόρος 4 = x y' z'. Μεγιστόρος 5 = x' + y + z'

Συνάρτηση σαν κανονικό άθροισμα

- Αφού κάθε γραμμή του πίνακα αληθείας αντιστοιχεί σε ένα ελαχιστόρο μπορούμε κοιτώντας το πίνακα αληθείας να δούμε ποιοι ελαχιστόροι επαληθεύουν τη συνάρτηση.
- Το λογικό άθροισμα αυτών μας δίνει την έκφραση της συνάρτησης σα κανονικό άθροισμα.

x	y	z	F_3
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- Η συνάρτηση επαληθεύεται όταν επαληθεύεται ο ελαχιστόρος 1 ή ο
  ελαχιστόρος 3 ή ο
  ελαχιστόρος 4 ή ο
  ελαχιστόρος 5

Ισοδύναμα είναι $F_3(x, y, z) = m_1 + m_3 + m_4 + m_5 =$
$x'y'z + x'yz + xy'z' + xy'z =$
$\Sigma(1, 3, 4, 5)$

- Αφού ισοδύναμες συναρτήσεις έχουν κοινό πίνακα αλήθειας, θα έχουν και κοινά κανονικά αθροίσματα. ***Συνεπώς το κανονικό άθροισμα είναι μια πρότυπη μορφή πάνω στην οποία μπορώ να εφαρμόσω κάποιον αλγόριθμο.***

Συνάρτηση σαν κανονικό γινόμενο

- Αφού κάθε γραμμή του πίνακα αληθείας αντιστοιχεί σε ένα μεγιστόρο μπορούμε κοιτώντας το πίνακα αληθείας να δούμε ποιοί μεγιστόροι **δεν** επαληθεύουν τη συνάρτηση.
- Το λογικό γινόμενο αυτών μας δίνει την έκφραση της συνάρτησης σα κανονικό γινόμενο.

x	y	z	F_3
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- Η συνάρτηση δεν επαληθεύεται αν επαληθεύεται (παίρνει δηλαδή τη τιμή 0) ο

μεγιστόρος 0 ή ο

μεγιστόρος 2 ή ο

μεγιστόρος 6 ή ο

μεγιστόρος 7

Ισοδύναμα είναι $F'_3(x, y, z) = M'_0 + M'_2 + M'_6 + M'_7 \Rightarrow$

$F_3(x, y, z) = M_0 \bullet M_2 \bullet M_6 \bullet M_7 =$

$(x + y + z) \bullet (x + y' + z) \bullet (x' + y' + z) \bullet (x' + y' + z') =$

$\Pi(0, 2, 6, 7)$

- Αφού ισοδύναμες συναρτήσεις έχουν κοινό πίνακα αλήθειας, θα έχουν και κοινά κανονικά γινόμενα. ***Συνεπώς το κανονικό γινόμενο είναι μια πρότυπη μορφή πάνω στην οποία μπορώ να εφαρμόσω κάποιον αλγόριθμο.***

Μετατροπές μεταξύ κανονικών μορφών

- Αφού κάθε γραμμή του πίνακα αληθείας είναι είτε 0 είτε 1 αν γνωρίζω τη μορφή της συνάρτησης στη μία κανονική μορφή η άλλη προκύπτει από τους όρους που λείπουν.
  - Π.χ. $F(A, B, C) = \Sigma\,(1,4,6) \Rightarrow F = \Pi\,(0, 2, 3, 5, 7)$

    $G(W, X, Y, Z) = \Pi\,(1, 8, 11, 14, 15) \Rightarrow$

    $G = \Sigma\,(0, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13)$
- Είναι επίσης πολύ εύκολο για μια συνάρτηση F να βρώ τη F'
- Ισχύει ότι $m'_i = M_i$ και φυσικά $M'_i = m_i$
  - Για παράδειγμα είναι $m'_0 = (x'\ y'\ z')' = x + y + z = M_0$
- Αρα αν $F(x, y, z) = \Sigma\,(1, 3,4) \Rightarrow F = m_1 + m_3 + m_4 \Rightarrow$

  $F' = (m_1 + m_3 + m_4)' = M_1 \bullet M_3 \bullet M_4 = \Pi(1, 3, 4) = \Sigma\,(0, 2, 5, 6, 7)$
- ***Δηλαδή η συμπληρωματική μιας συνάρτησης προκύπτει σα κανονικό άθροισμα των ελαχιστόρων που λείπουν από το κανονικό της άθροισμα***

Να θυμάστε

Πίνακας αλήθειας
Κανονικό άθροισμα
Κανονικό γινόμενο

} πρότυπες μορφές

Αλγεβρική μορφή
Λογικό διάγραμμα
Λεκτική περιγραφή

} μή πρότυπες μορφές

Πρέπει επίσης να μπορείτε να πηγαίνετε από μη πρότυπες μορφές σε πρότυπες.

Απλοποίηση Συναρτήσεων

- Η πολυπλοκότητα του κυκλώματος που υλοποιεί μια συνάρτηση Boole σχετίζεται άμεσα με την πολυπλοκότητα της αλγεβρικής έκφρασης από την οποία η συνάρτηση υλοποιείται.
- Σκοποί της απλοποίησης
  - Λιγότεροι όροι
  - Απλούστεροι όροι
- Θέλουμε απλές και συστηματικές μεθόδους
- Υπάρχουν :
  - Η μέθοδος του χάρτη (μέθοδος Karnaugh / k-map) : γραφική μέθοδος για συναρτήσεις έως 5 μεταβλητών.
  - Η μέθοδος Quine-McClauskey : αλγεβρική μέθοδος
  - Η μέθοδος Espresso : αλγεβρική μέθοδος
- ***Οι μέθοδοι αυτοί δε μας δίνουν τις υλοποιήσεις με τις λιγότερες πύλες, αλλά τις απλούστερες υλοποιήσεις με NOT, AND & OR.***

Η Μέθοδος του Χάρτη

- Ο χάρτης είναι ένα διάγραμμα αποτελούμενο από τετράγωνα.
- Κάθε τετράγωνο παριστάνει ένα ελαχιστόρο.
- Αν ο ελαχιστόρος αληθεύει τη συνάρτηση, υπάρχει 1 στο αντίστοιχο τετράγωνο. Αν όχι υπάρχει 0.
- "Γειτονιές" (Τετράγωνα / ορθογώνια 2 ή 4 ή 8 ή 16 ή … επαληθευόμενων ή αδιάφορων ελαχιστόρων) στο χάρτη υποδεικνύουν ελαχιστόρους που μπορούν να απλοποιηθούν

Karnaugh 2-Μεταβλητών

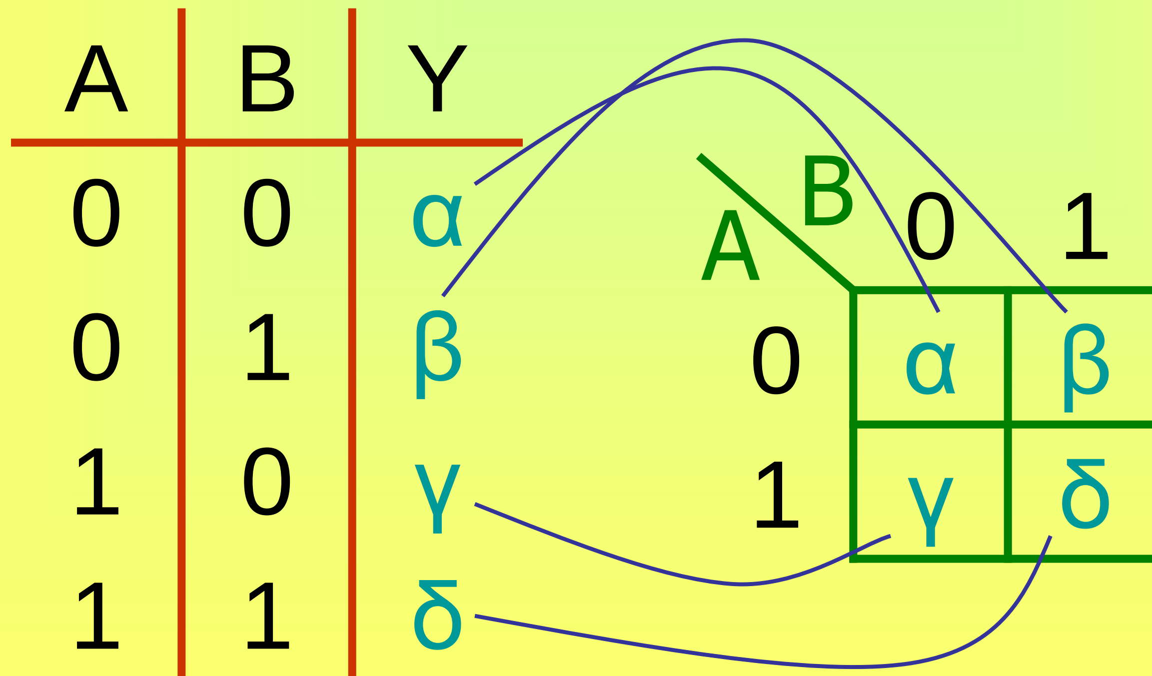
A	B	Y
0	0	α
0	1	β
1	0	γ
1	1	δ

Karnaugh 2-Μεταβλητών

A	B	Y
0	0	α
0	1	β
1	0	γ
1	1	δ

A \ B	0	1
0		
1		

Karnaugh 2-Μεταβλητών



A	B	Y
0	0	α
0	1	β
1	0	γ
1	1	δ

$\overline{B}$

A \ B	0	1
0	α	β
1	γ	δ

$\overline{B}$

A \ B	0	1
0	α	β
1	γ	δ

B

A \ B	0	1
0	α	β
1	γ	δ

$\overline{B}$

A \ B	0	1
0	α	β
1	γ	δ

B

A \ B	0	1
0	α	β
1	γ	δ

$\overline{A}$

A \ B	0	1
0	α	β
1	γ	δ

$\overline{B}$

A \ B	0	1
0	α	β
1	γ	δ

B

A \ B	0	1
0	α	β
1	γ	δ

$\overline{A}$

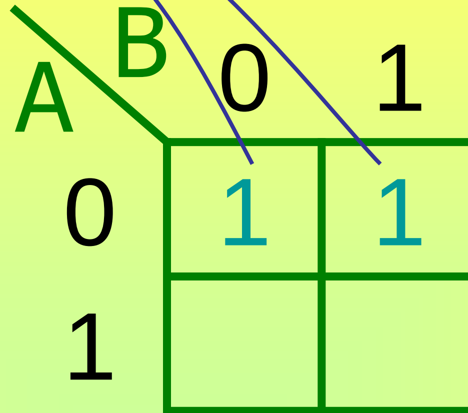
A \ B	0	1
0	α	β
1	γ	δ

A

A \ B	0	1
0	α	β
1	γ	δ

A	B	Y
0	0	1
0	1	1
1	0	0
1	1	0

A	B	Y
0	0	1
0	1	1
1	0	0
1	1	0



A	B	Y
0	0	1
0	1	1
1	0	0
1	1	0

A \ B	0	1
0	1	1
1		

$$Y = \overline{A}$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

A \ B	0	1
0		1
1	1	1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

A \ B	0	1
0		1
1	1	1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

A \ B	0	1
0		1
1	1	1

$$Y = A + B$$

A \ B	0	1
0		1
1	1	

$$Y = \overline{A}B + A\overline{B} = A \oplus B$$

Χάρτης Δυο (2) Μεταβλητών

Ο χάρτης περιέχει 4 τετράγωνα, ένα για κάθε ελαχιστόρο.

m_0	m_1
m_2	m_3

x \ y	0	1
0	x΄y΄	x΄y
1	xy΄	xy

Το x εμφανίζεται ως συμπλήρωμα στη γραμμή 0 και κανονικά στη γραμμή 1.

Το y εμφανίζεται ως συμπλήρωμα στη στήλη 0 και κανονικά στη στήλη 1.

Παραδείγματα

x \ y	0	1
0		
1		1

$xy = \Sigma(3) = m_3$

x \ y	0	1
0		1
1	1	1

$x+y = \Sigma(1,2,3) = m_1 + m_2 + m_3$

Karnaugh 3-Μεταβλητών

A \ BC	00	01	11	10
0				
1				

- **Προσοχή**
  - Στη σειρά *00 01 11 10*
  - Κώδικας *Gray*
  - Τα γειτονικά κελιά διαφέρουν κατά 1 bit ή μια μεταβλητή

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$$

000 001

A \ BC	00	01	11	10
0	1	1		
1				

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$$

000 001

A \ BC	00	01	11	10
0	1	1		
1				

$$Y = \bar{A}\bar{B}$$

A \ BC	00	01	11	10
0	1	1	1	1
1				

A \ BC	00	01	11	10
0	1	1	1	1
1				

$$Y = \overline{A}$$

A \ BC	00	01	11	10
0	1	1	1	1
1				

A \ BC	00	01	11	10
0		1	1	
1		1	1	

$$Y = \overline{A}$$

A \ BC	00	01	11	10
0	1	1	1	1
1				

$$Y = \overline{A}$$

A \ BC	00	01	11	10
0		1	1	
1		1	1	

$$Y = C$$

A \ BC	00	01	11	10
0	1	1	1	1
1				

$$Y = \overline{A}$$

A \ BC	00	01	11	10
0		1	1	
1		1	1	

$$Y = C$$

A \ BC	00	01	11	10
0	1	1	1	1
1			1	1

A \ BC	00	01	11	10
0	1	1	1	1
1				

$$Y = \overline{A}$$

A \ BC	00	01	11	10
0		1	1	
1		1	1	

$$Y = C$$

A \ BC	00	01	11	10
0	1	1	1	1
1			1	1

$$Y = \overline{A} + B$$

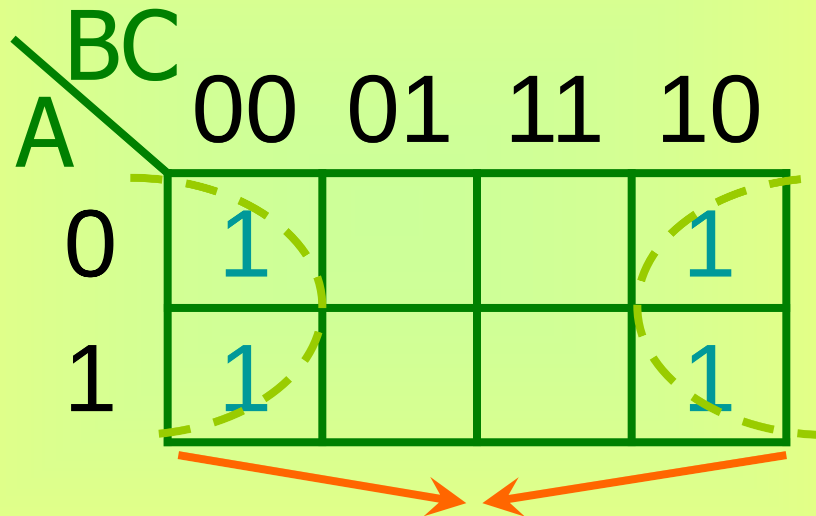
$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + AB\overline{C}$$

000 100 010 110

A \ BC	00	01	11	10
0	1			1
1	1			1

$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + AB\overline{C}$$

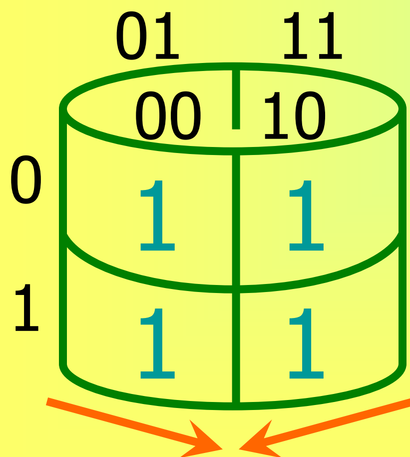
000 100 010 110



$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C}$$

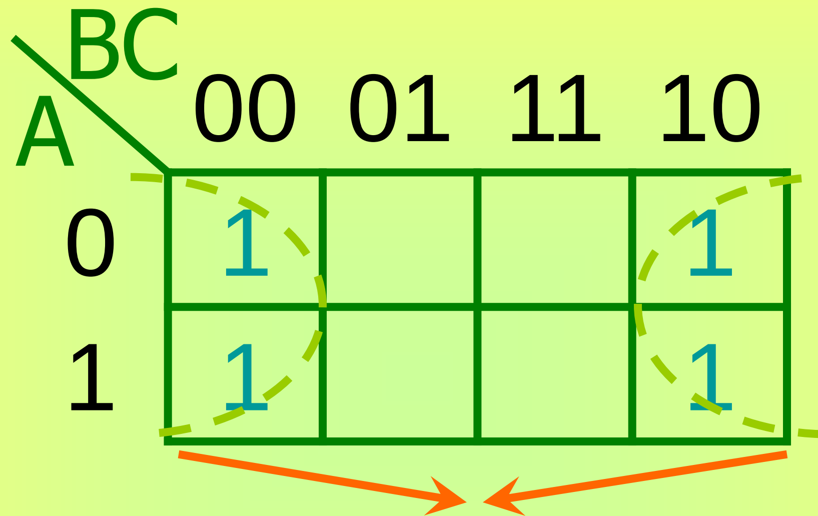
000 100 010 110

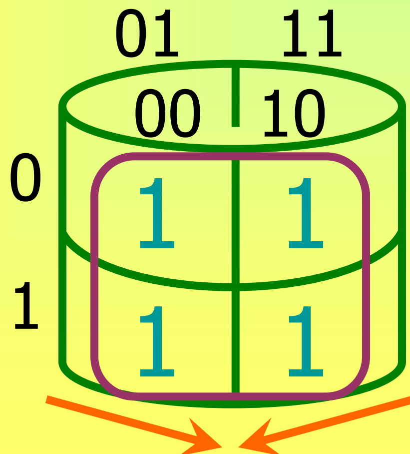
A \ BC	00	01	11	10
0	1			1
1	1			1



$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C}$$

000 100 010 110



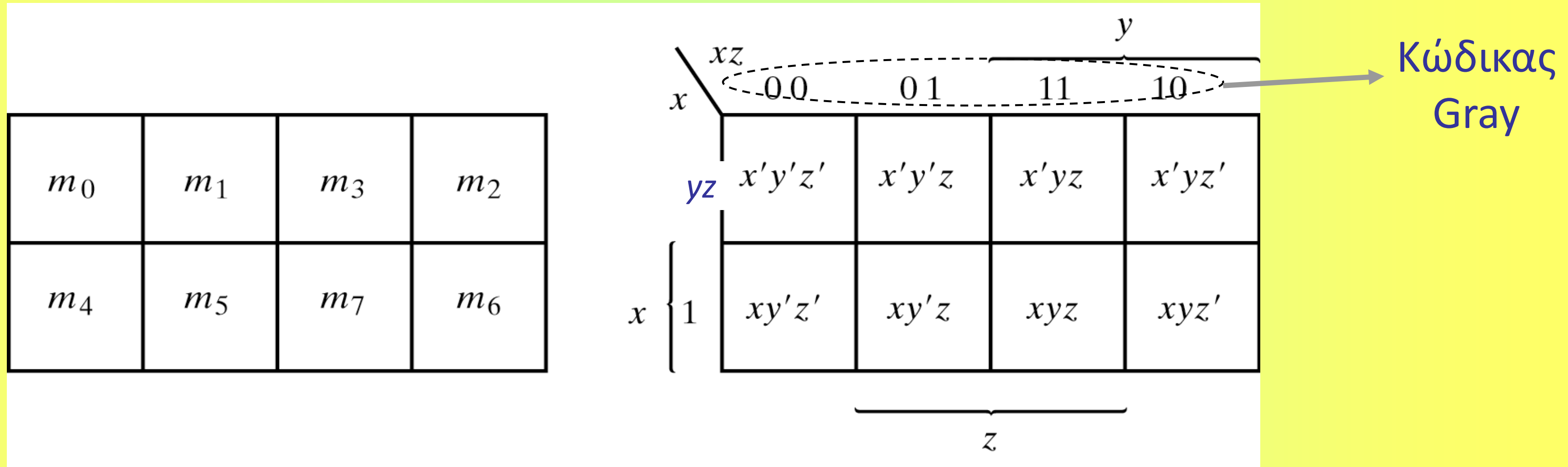


$$Y = \bar{C}$$

Χάρτης Τριών (3) Μεταβλητών

Ο χάρτης περιέχει 8 τετράγωνα, ένα για κάθε ελαχιστόρο.

Οι ελαχιστόροι τοποθετούνται σε σειρά όμοια με τον κώδικα Gray.



- Το άθροισμα δύο ελαχιστόρων της συνάρτησης που βρίσκονται σε γειτονικά τετράγωνα απλοποιείται σε ένα όρο ΚΑΙ με δύο μόνο παράγοντες.
- Το άθροισμα τεσσάρων ελαχιστόρων της συνάρτησης που βρίσκονται σε γειτονικά τετράγωνα απλοποιείται σε ένα όρο με ένα μόνο παράγοντα.
- Το άθροισμα οκτώ ελαχιστόρων της συνάρτησης που βρίσκονται σε γειτονικά τετράγωνα καταλαμβάνει όλο το χάρτη και παριστάνει τη συνάρτηση που είναι πάντα ίση με 1.

Παραδείγματα Χάρτη Τριών (3) Μεταβλητών

$F(x,y,z) = \Sigma(2,3,4,5)$

x \ yz	00	01	11	10
0			1	1
1	1	1		

$F(x,y,z) = x'y+xy'$

$F(x,y,z) = \Sigma(3,4,6,7)$

x \ yz	00	01	11	10
0			1	
1	1		1	1

$F(x,y,z) = yz+xz'$

Παραδείγματα Χάρτη Τριών (3) Μεταβλητών

$F(x,y,z) = \Sigma(0,2,4,5,6)$

x \ yz	00	01	11	10
0	1			1
1	1	1		1

$F(x,y,z) = z' + xy'$

$F(A,B,C) =$
$A'C + A'B + AB'C + BC$

A \ BC	00	01	11	10
0		1	1	1
1		1	1	

$F(A,B,C) = \Sigma(1,2,3,5,7) = C + A'B$

Karnaugh 4-Μεταβλητών

AB \ CD	00	01	11	10
00				
01				
11				
10				

AB \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	1
10			1	

AB \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	1
10			1	

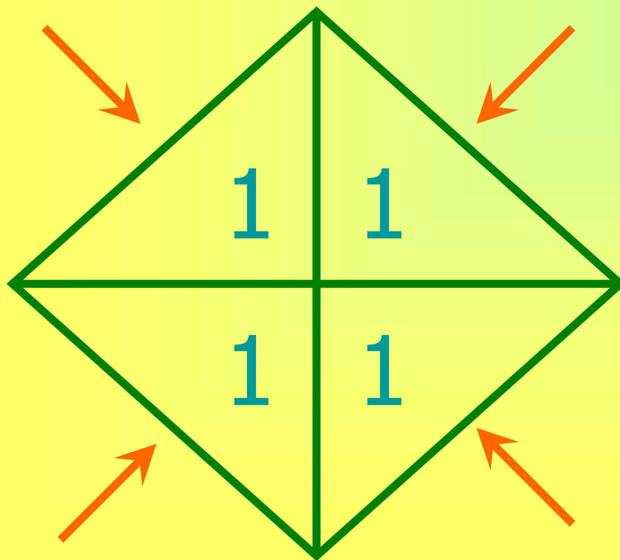
AB \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	1
10			1	

AB \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	1
10			1	

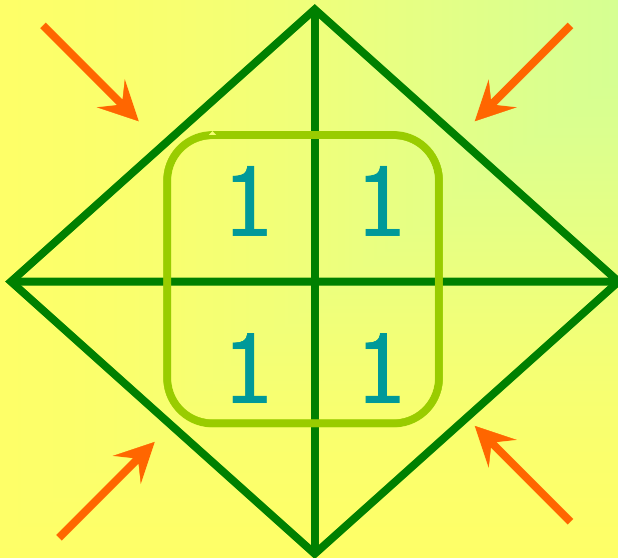
AB \ CD	00	01	11	10
00			1	
01			1	
11	1	1	1	1
10			1	

$$Y = AB + CD$$

AB \ CD	00	01	11	10
00	1			1
01				
11				
10	1			1



AB \ CD	00	01	11	10
00	1			1
01				
11				
10	1			1



$$Y = \overline{B}\,\overline{D}$$

AB \ CD	00	01	11	10
00	1	1	1	1
01				
11				
10	1	1	1	1

$$Y = \overline{B}$$

AB \ CD	00	01	11	10
00	1	1	1	1
01	1			1
11	1			1
10	1	1	1	1

$$Y = \overline{B} + \overline{D}$$

AB \ CD	00	01	11	10
00	1		1	
01	1		1	
11	1	1	1	
10	1		1	

$$Y = \overline{C}\,\overline{D} + CD + AB\overline{C}$$

Χάρτης Τεσσάρων (4) Μεταβλητών

Ο χάρτης περιέχει 16 τετράγωνα, ένα για κάθε ελαχιστόρο.
Οι ελαχιστόροι τοποθετούνται σε σειρά όμοια με τον κώδικα Gray.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

wx \ yz	00	01 (z)	11 (y, z)	10 (y)
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01 (x)	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11 (w, x)	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10 (w)	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

Κάθε 2^n γειτονικά τετράγωνα διαφέρουν σε n μεταβλητές και οδηγούν σε έναν όρο ΚΑΙ με $k – n$ παράγοντες, όπου k το πλήθος των μεταβλητών της συνάρτησης.

Η πάνω ακμή ακουμπάει στην κάτω και η δεξιά στην αριστερή (γειτονικότητα).

Παραδείγματα Χάρτη Τεσσάρων (4) Μεταβλητών

$F(w,x,y,z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

$F(w,x,y,z) = y' + w'z' + xz'$

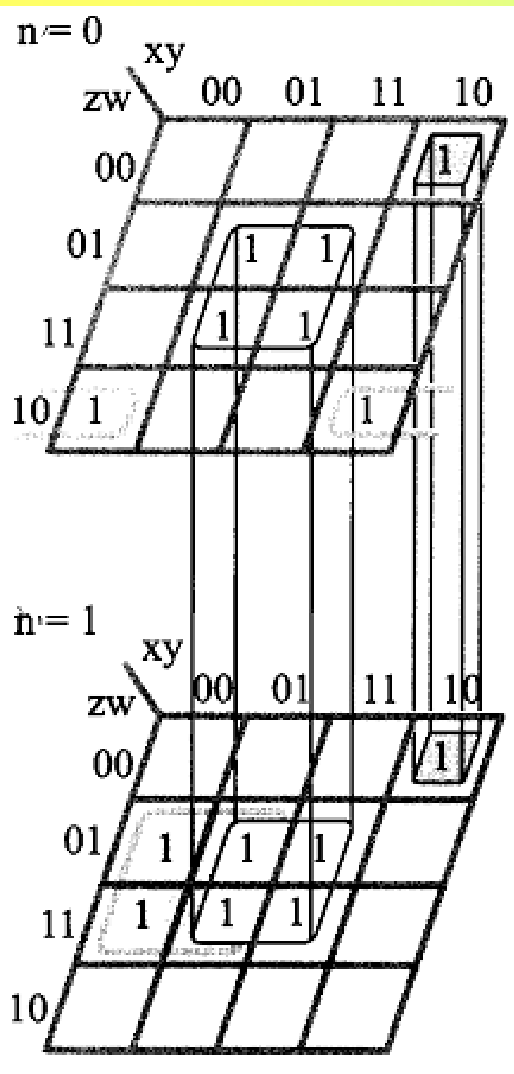
wx \ yz	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

AB \ CD	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

$F(A,B,C,D) = A'B'C' + B'CD' + A'BCD' + AB'C'$

$F(A,B,C,D) = B'D' + B'C' + A'CD'$

Χάρτης Πέντε (5) μεταβλητών



A = 0

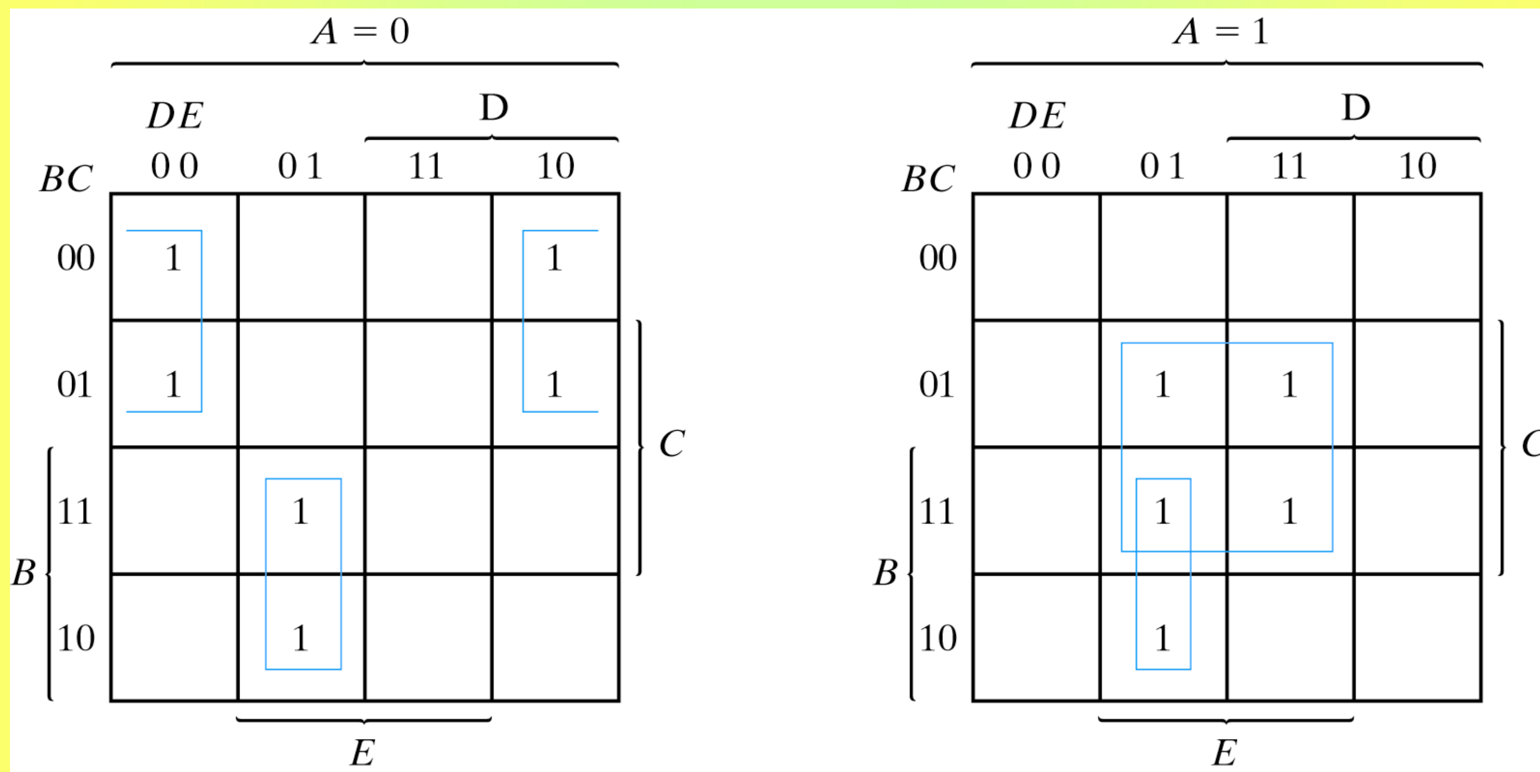
BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

(D: columns 11, 10; E: columns 01, 11; C: rows 01, 11; B: rows 11, 10)

Παράδειγμα Χάρτη Πέντε (5) Μεταβλητών



F(A, B, C, D, E) = Σ(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)

F(A, B, C, D, E) = A΄B΄E΄ + BD΄E + ACE

Συνθήκες Αδιαφορίας

Συμβολίζονται με × και αντιστοιχούν σε συνδυασμούς εισόδων που δεν ορίζονται για μία συνάρτηση.

Π.χ. F(w,x,y,z)=Σ(1,3,7,11,15) με συνθήκες αδιαφορίας d(w,x,y,z)=Σ(0,2,5)

wx \ yz	00	01	11	10
00	×	1	1	×
01	0	×	1	0
11	0	0	1	0
10	0	0	1	0

F = yz + w΄x΄ = Σ(0,1,2,3,7,11,15)

wx \ yz	00	01	11	10
00	×	1	1	×
01	0	×	1	0
11	0	0	1	0
10	0	0	1	0

F = yz + w΄z = Σ(1,3,5,7,11,15)

Οι αδιάφοροι όροι μπορούν να χρησιμοποιηθούν ως άσσοι ή μηδενικά ανάλογα με την απλοποίηση που οδηγεί στο μικρότερο κύκλωμα.

ABCD	
0000	X
0001	1
0010	1
0011	1
0100	1
0101	0
0110	X
0111	1
1000	1
1001	1
1010	1
1011	1
1100	0
1101	0
1110	X
1111	1

AB \ CD	00	01	11	10
00	X	1	1	1
01	1	0	1	X
11	0	0	1	X
10	1	1	1	1

ABCD	
0000	X
0001	1
0010	X
0011	1
0100	0
0101	X
0110	1
0111	0
1000	1
1001	0
1010	X
1011	1
1100	X
1101	0
1110	0
1111	0

AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	0	1
11	X	0	0	0
10	1	0	1	X

ABCD	
0000	0
0001	1
0010	0
0011	1
0100	1
0101	1
0110	0
0111	1
1000	X
1001	1
1010	1
1011	0
1100	1
1101	0
1110	X
1111	X

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	1	0	X	X
10	X	1	0	1

ABCD	
0000	1
0001	1
0010	1
0011	X
0100	1
0101	1
0110	X
0111	X
1000	1
1001	0
1010	1
1011	X
1100	0
1101	0
1110	X
1111	1

AB \ CD	00	01	11	10
00	1	1	X	1
01	1	1	X	X
11	0	0	1	X
10	1	0	X	1

Πρωτεύοντες Όροι (Prime Implicants)

Πρωτεύοντας Όρος (πρώτος συνεπαγωγός ή prime implicant ή PI) :

ένα γινόμενο παραγόντων που σχηματίζεται συνδυάζοντας το μεγαλύτερο δυνατό αριθμό γειτονικών τετραγώνων.

Θεμελιώδης Ορος (ουσιώδης πρώτος συνεπαγωγός) :

όταν καλύπτει ένα ελαχιστόρο που δεν καλύπτει κανένας άλλος prime implicant.

F(A, B, C, D)=Σ(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

C, B, A, D

Θεμ. PI: BD, B΄D΄

AB \ CD	00	01	11	10
00	1		1	1
01		1	1	
11		1	1	
10	1	1	1	1

C, B, A, D

PI: CD, B΄C, AD, AB΄

m_3: CD, B΄C

m_9: AD, AB΄

m_{11}: CD, B΄C, AD, AB΄

F=(BD+B΄D΄)+
(CD+AD) or (CD + AB΄) or (B΄C+AD) or (B΄C+AB΄)

Αλγόριθμος απλοποίησης με πίνακα Karnaugh

- Βήμα 1 :
  - Βρες όλους τους ουσιώδεις πρώτους συνεπαγωγούς της συνάρτησης
    - Για κάθε 1 του πίνακα βρες τις "γειτονιές του"
    - Επέλεξε τις μέγιστες σε πλήθος γειτονιές. Αυτοί είναι οι πρώτοι συνεπαγωγοί
    - Επέλεξε από τους πρώτους συνεπαγωγούς τους μοναδικούς που καλύπτουν κάποιο 1. Αυτοί είναι οι ουσιώδεις πρώτοι συνεπαγωγοί
- Βήμα 2 :
  - Για κάθε 1 του πίνακα που δεν έχει ήδη "καλυφθεί", επέλεξε τυχαία ένα πρώτο συνεπαγωγό του
- Βήμα 3 :
  - Πήγαινε στο Βήμα 2 μέχρι να "καλυφθούν" όλοι οι 1 του πίνακα.

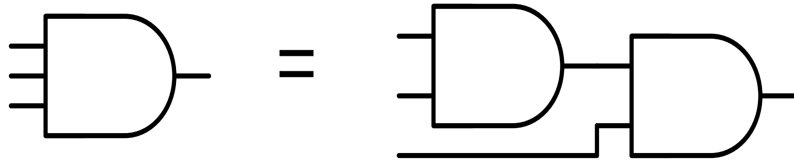
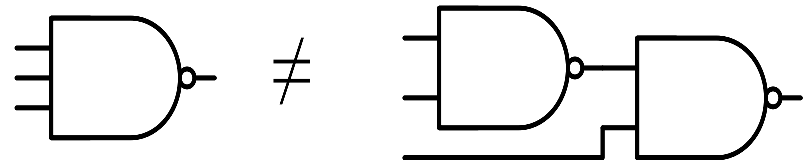
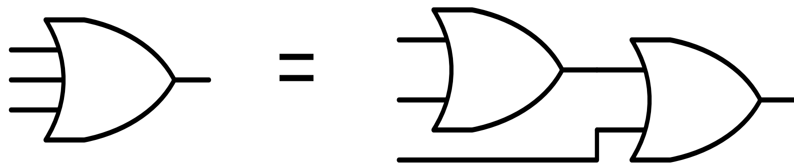
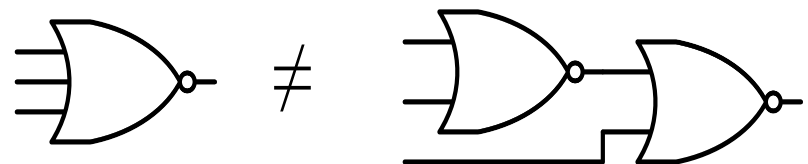
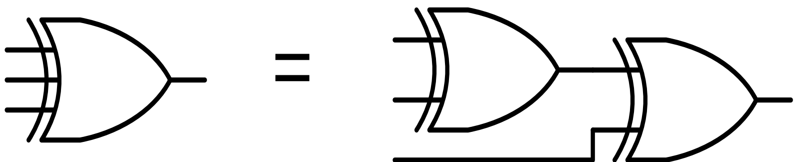
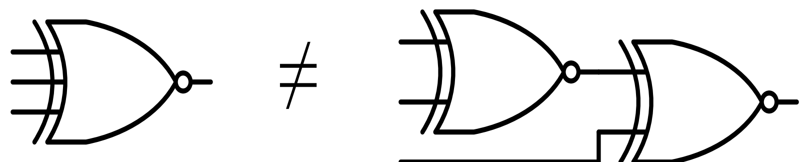
Ετσι έχουμε για τα λογικά μας διαγράμματα : (1/2)

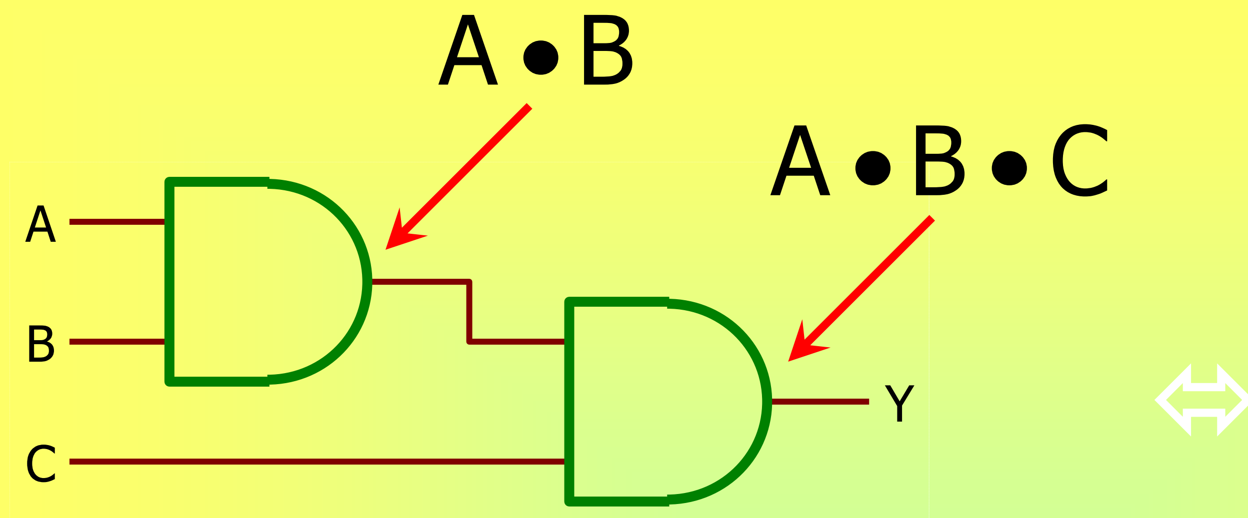
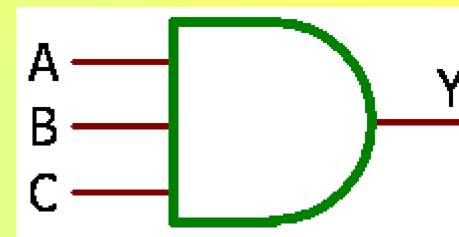
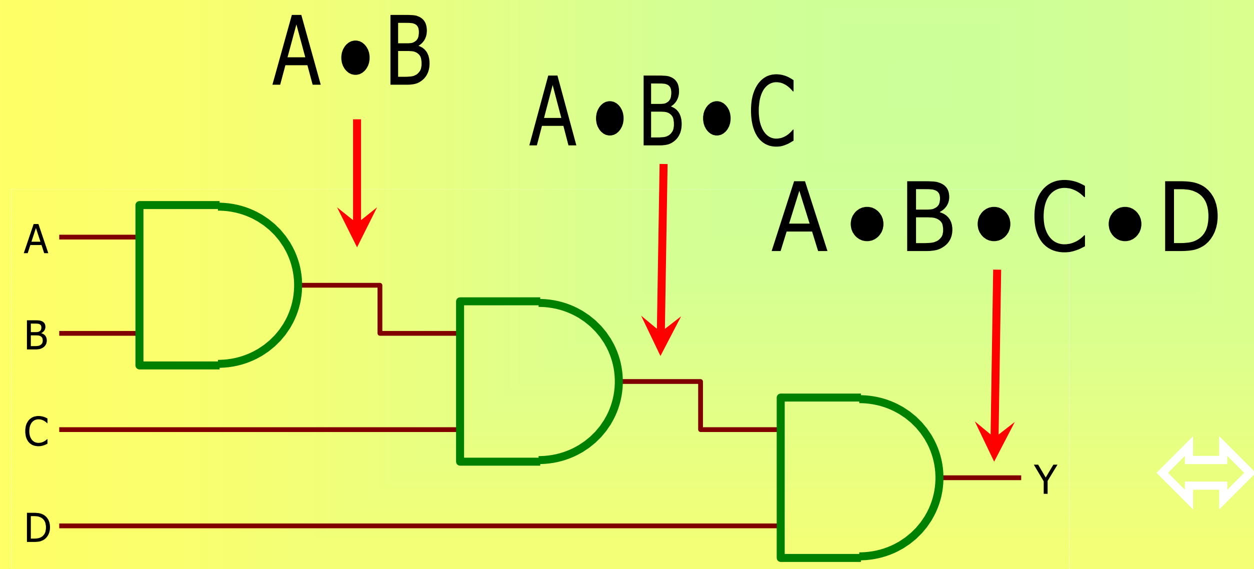
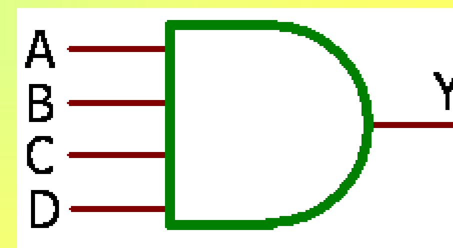
Όνομα	Γραφικό Σύμβολο	Αλγεβρική Συνάρτηση	Πίνακας Αληθείας
AND ΚΑΙ	x, y → (symbol) → F	$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR Ή	x, y → (symbol) → F	$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Αντιστροφέας	x → (symbol) → F	$F = x'$	x | F 0 | 1 1 | 0
Απομονωτής	x → (symbol) → F	$F = x$	x | F 0 | 0 1 | 1

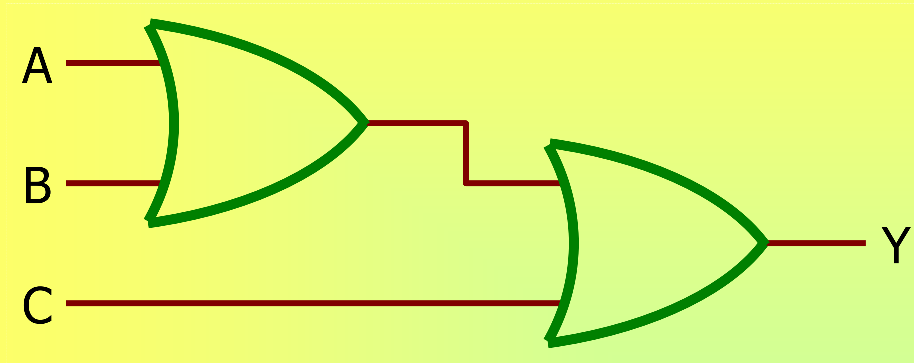
Ετσι έχουμε για τα λογικά μας διαγράμματα : (2/2)

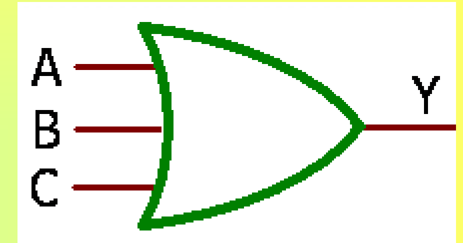
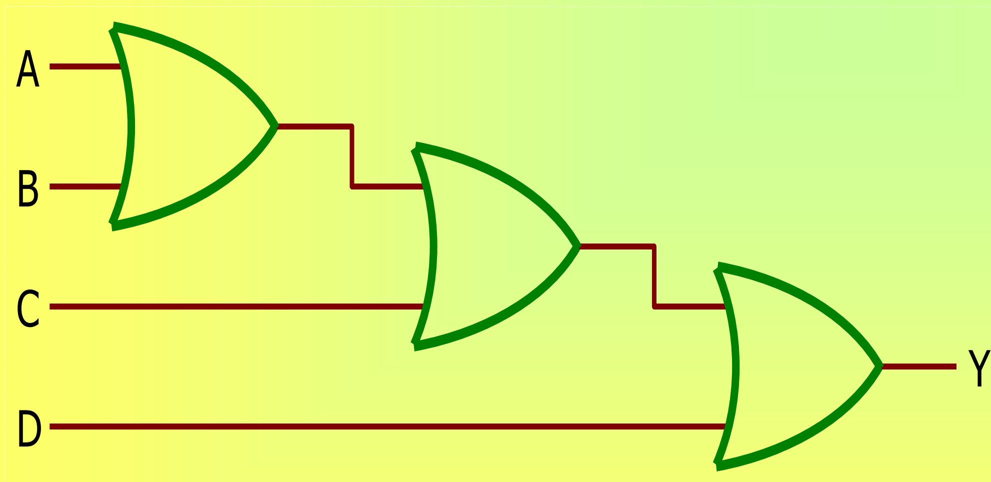
Όνομα	Γραφικό Σύμβολο	Αλγεβρική Συνάρτηση	Πίνακας Αληθείας
NAND ΟΧΙ-ΚΑΙ	x, y → F	$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR ΟΥΤΕ	x, y → F	$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR Αποκλειστό - Ή	x, y → F	$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Ισοδυναμία ή Αποκλειστικό -ΟΥΤΕ	x, y → F	$F = xy + x'y'$ $= x \odot y$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

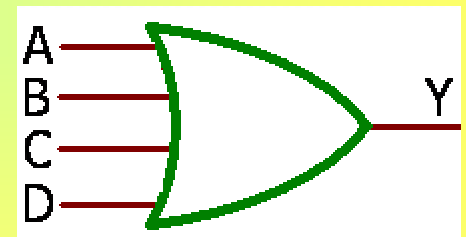
Επέκταση σε περισσότερες εισόδους ? (2)

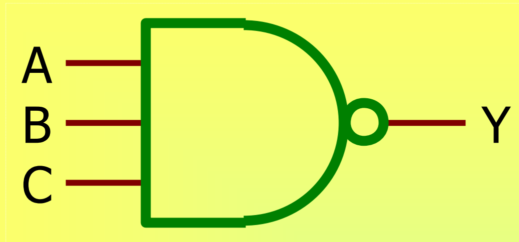








A • B
A • B • C
A
B
C
Y

A
B
C
Y

A • B
A • B • C
A • B • C • D
A
B
C
D
Y

A
B
C
D
Y


A
B
C
Y

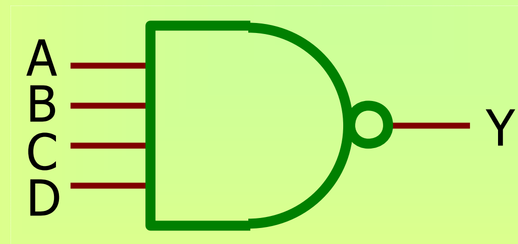

A
B
C
Y

A
B
C
D
Y


A
B
C
D
Y



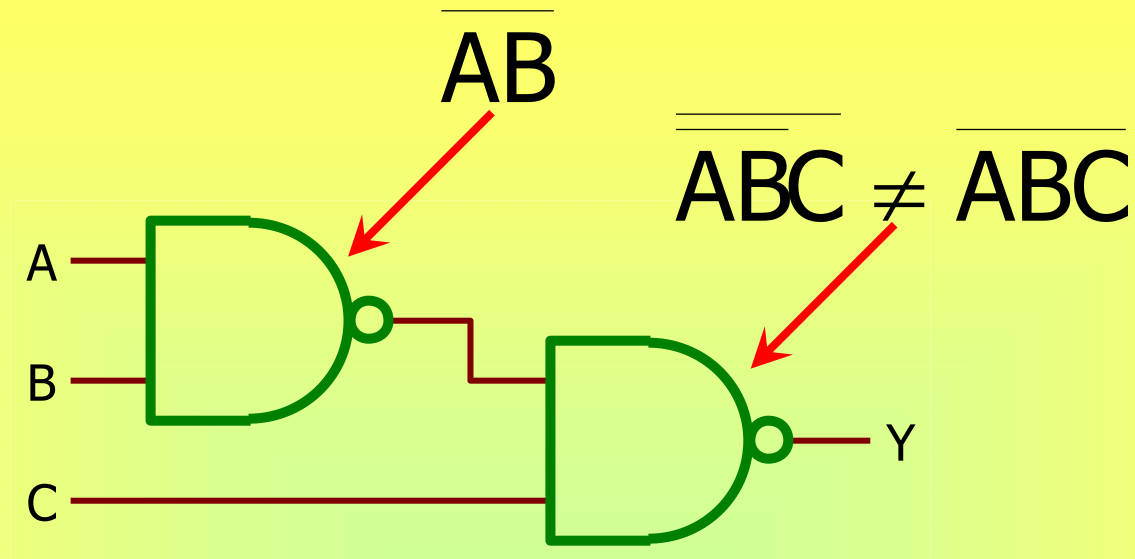
$$Y = \overline{A \bullet B \bullet C}$$

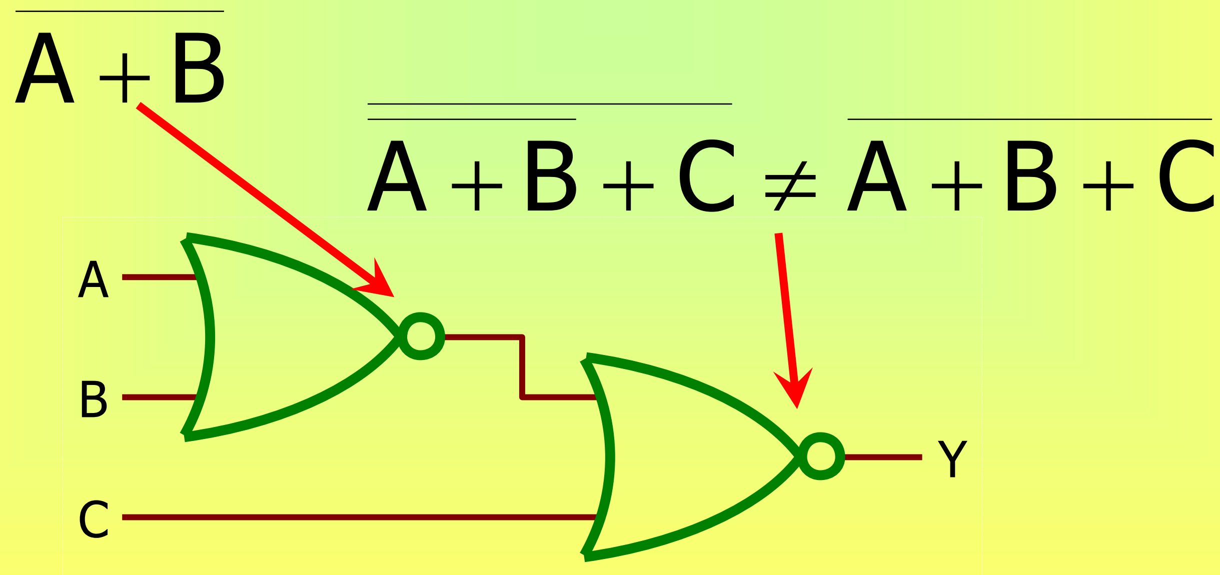
A	B	C	Y
X	X	0	1
X	0	X	1
0	X	X	1
1	1	1	0

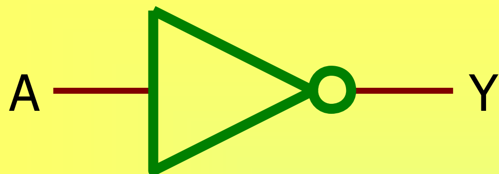
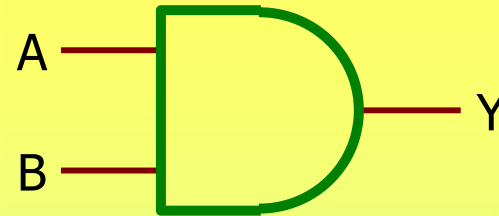
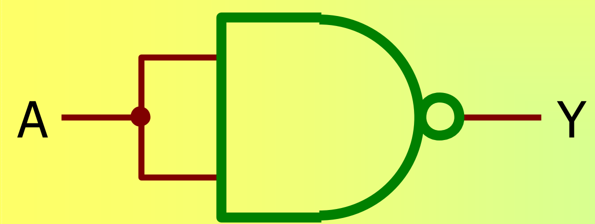
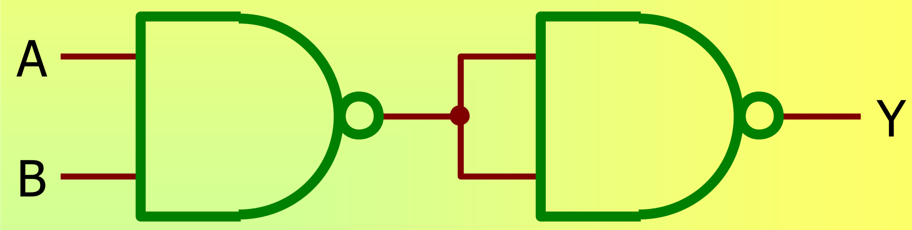
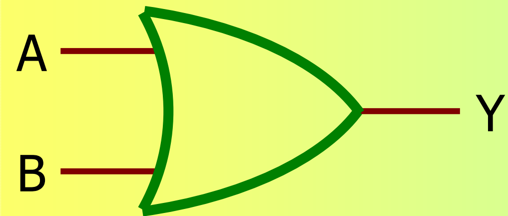
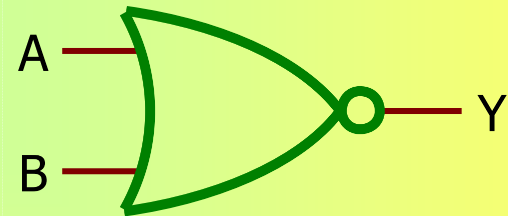
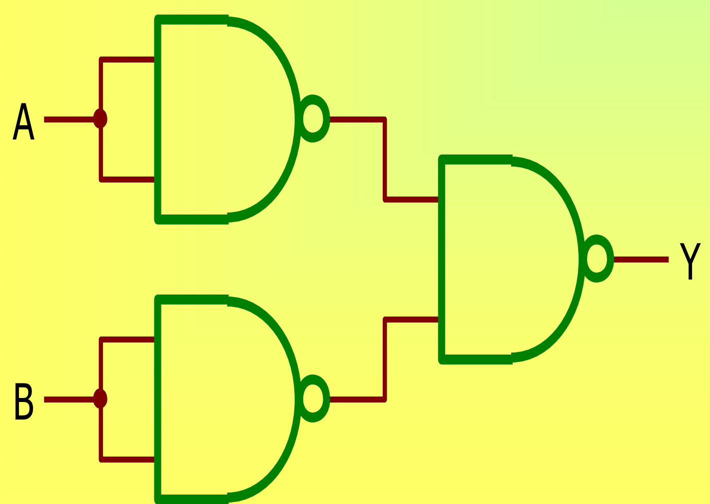
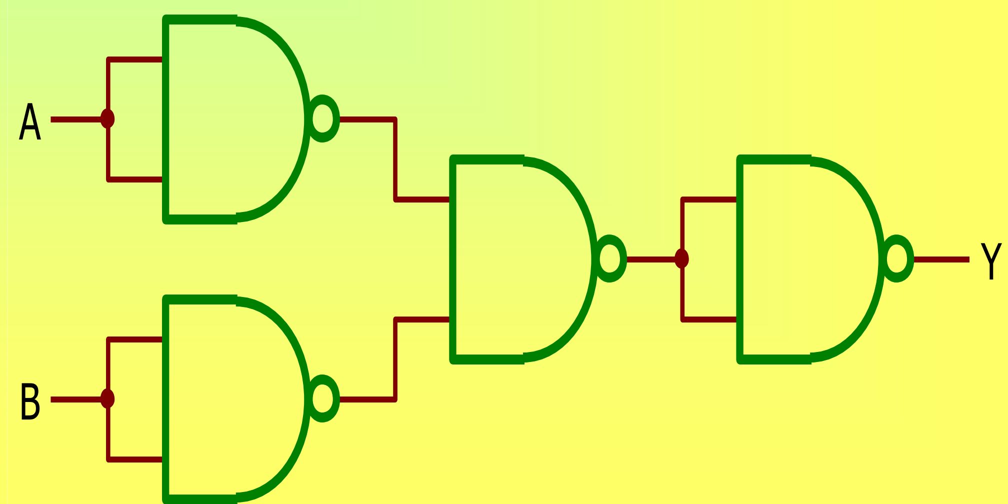


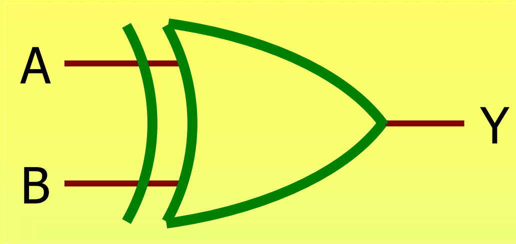
$$Y = \overline{A \bullet B \bullet C \bullet D}$$

A	B	C	D	Y
X	X	X	0	1
X	X	0	X	1
X	0	X	X	1
0	X	X	X	1
1	1	1	1	0

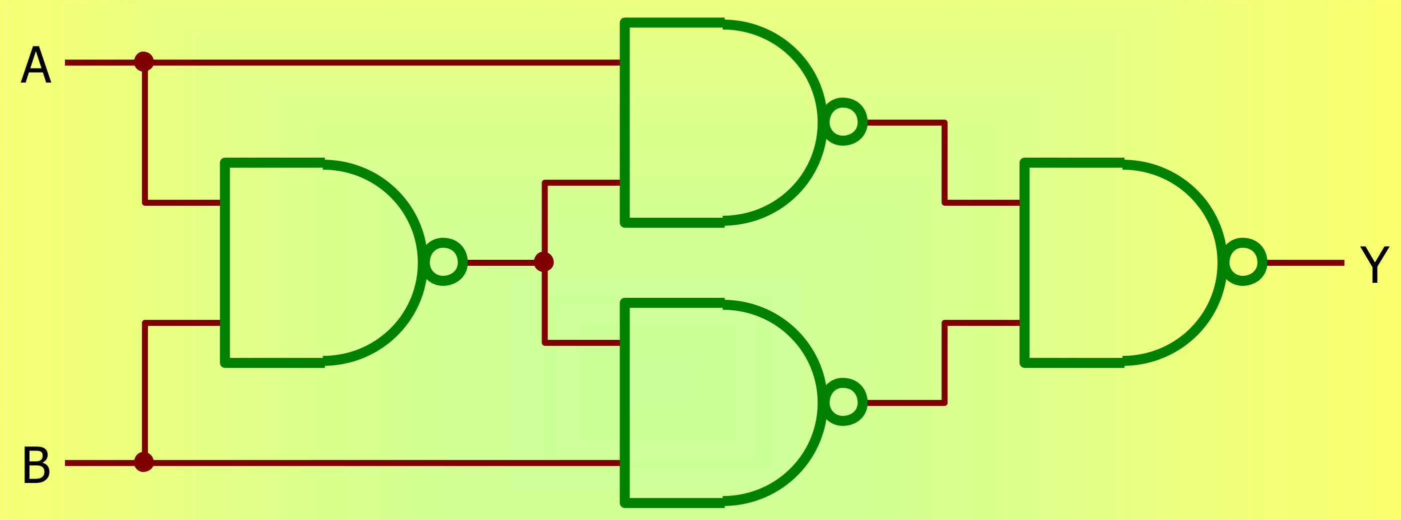

$\overline{AB}$
$\overline{\overline{AB}C} \neq \overline{ABC}$
A
B
C
Y


$\overline{A+B}$
$\overline{\overline{A+B}+C} \neq \overline{A+B+C}$
A
B
C
Y


A
Y

A
B
Y

A
Y

A
B
Y

A
B
Y

A
B
Y

A
B
Y

A
B
Y



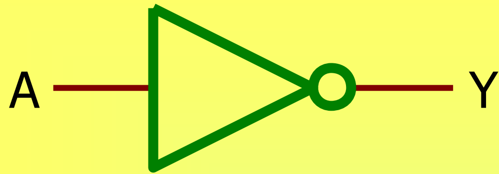
$$Y = A \oplus B$$

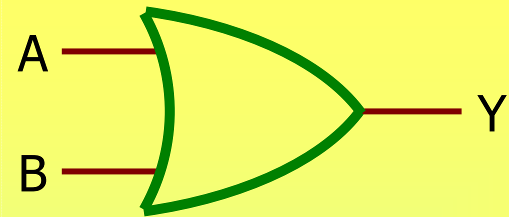


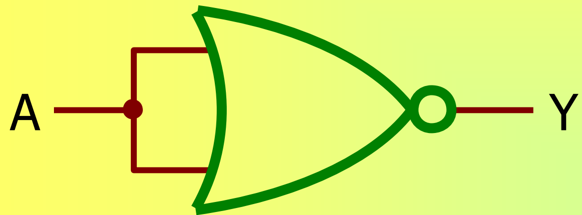
$$Y = \overline{\overline{A\overline{AB}}\;\overline{B\overline{AB}}} = A\overline{AB} + B\overline{AB} =$$

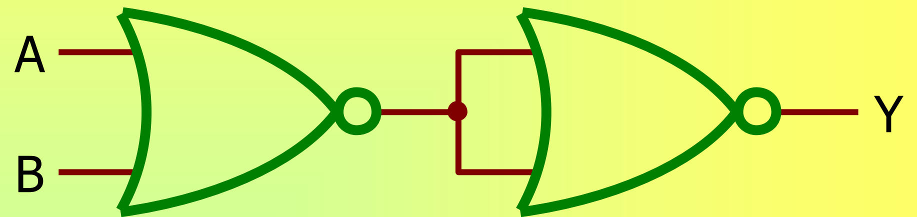
$$= A\left(\overline{A} + \overline{B}\right) + B\left(\overline{A} + \overline{B}\right) =$$

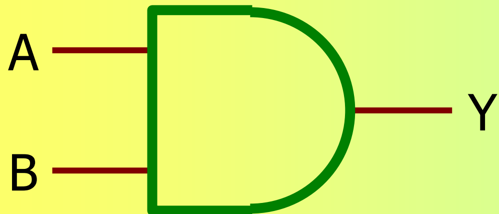
$$= \overline{A}B + A\overline{B} = A \oplus B$$

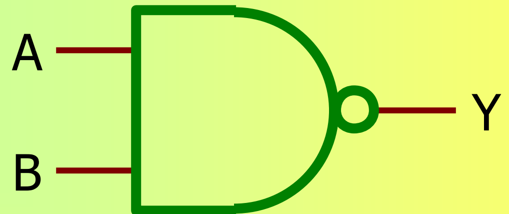

A
Y

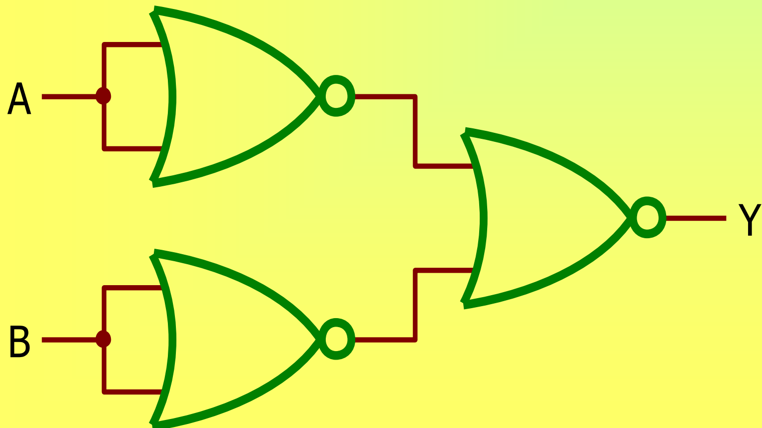

A
B
Y

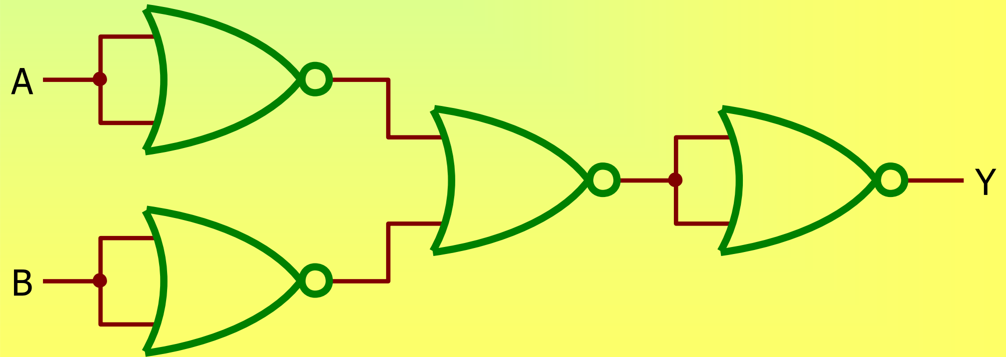

A
Y


A
B
Y

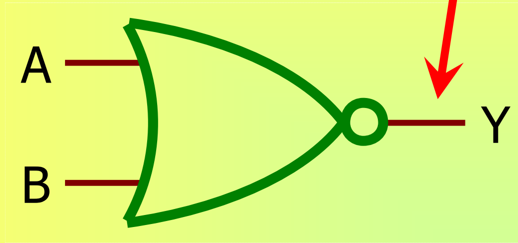

A
B
Y

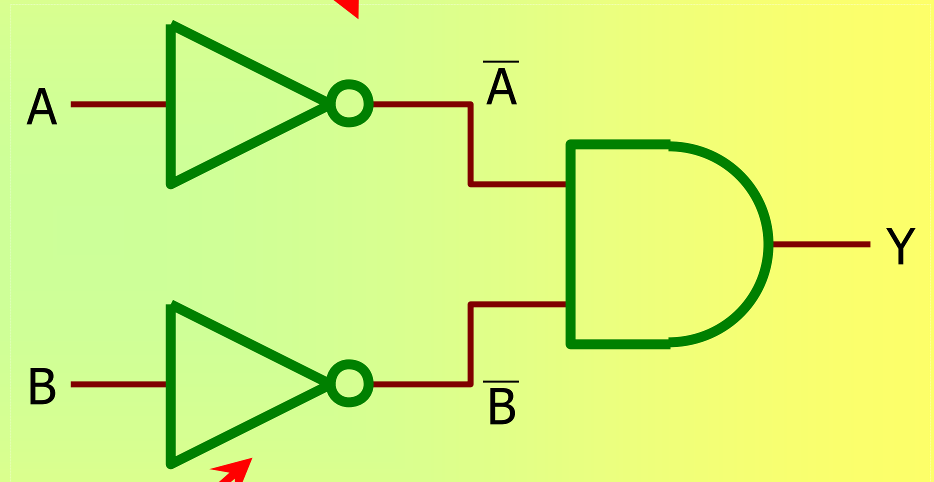

A
B
Y

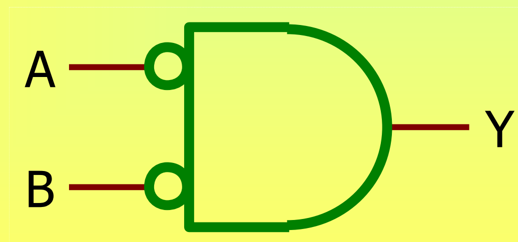

A
B
Y


A
B
Y

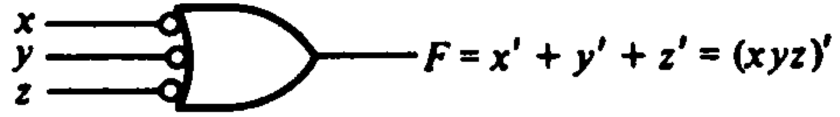
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$



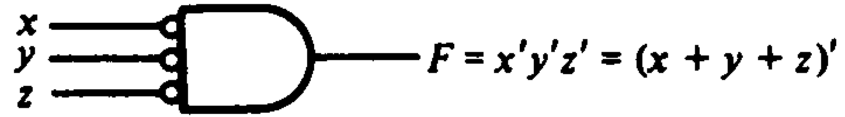




Υλοποίηση με πύλες NAND & NOR



(α) Δύο σύμβολα για πύλες ΟΧΙ-ΚΑΙ



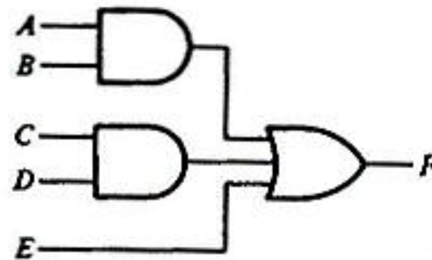
(β) Δύο σύμβολα για πύλες ΟΥΤΕ

(γ) Τρία σύμβολα για αντιστροφείς

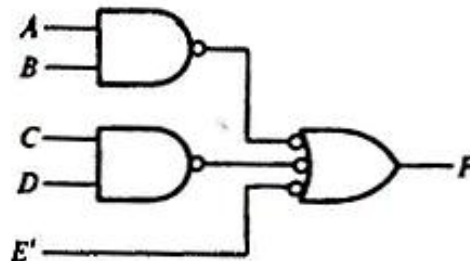
Υλοποίηση με πύλες NAND

<u>Παράδειγμα:</u>

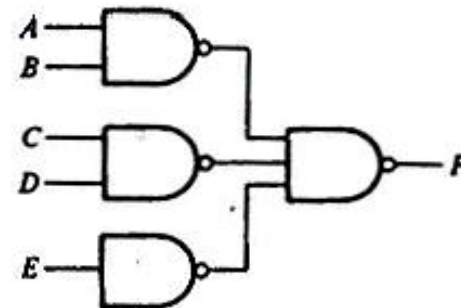
Υλοποίηση της συνάρτησης F=AB+CD+E με NAND.



(α) ΚΑΙ-Ή (AND-OR)

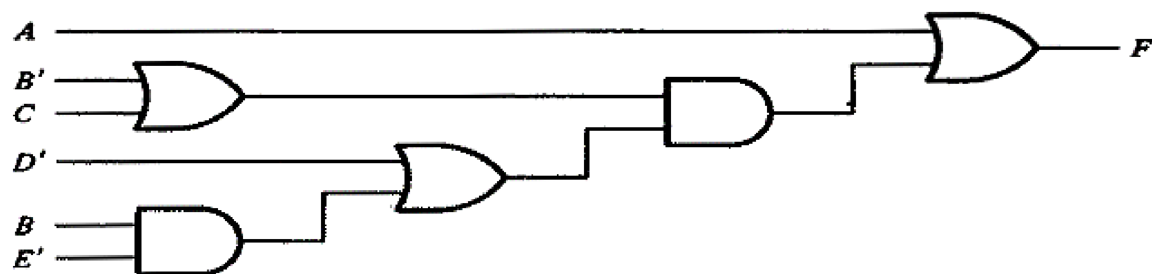


(β) ΟΧΙ ΚΑΙ-ΟΧΙ ΚΑΙ
(NAND-NAND)

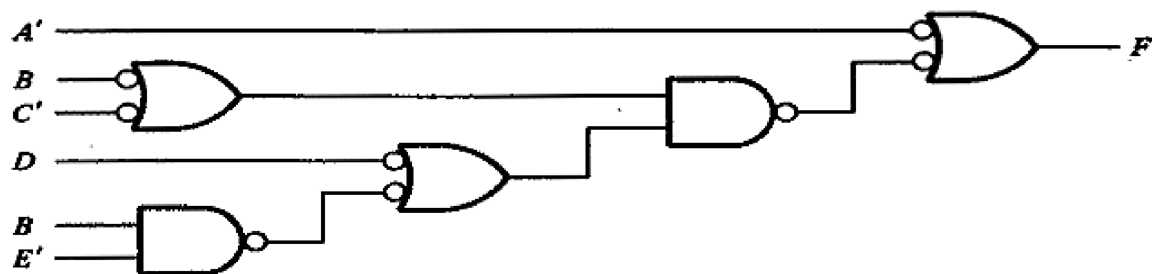


(γ) ΟΧΙ ΚΑΙ-ΟΧΙ ΚΑΙ
(NAND-NAND)

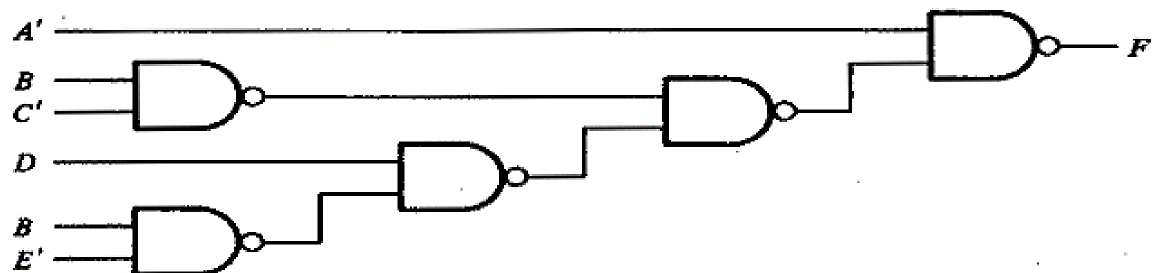
Παράδειγμα (1)



(α) Διάγραμμα ΚΑΙ-Ή



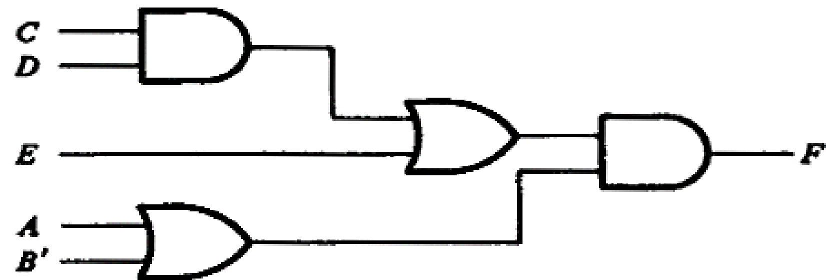
(β) Διάγραμμα με πύλες ΟΧΙ-ΚΑΙ με δύο γραφικά σύμβολα



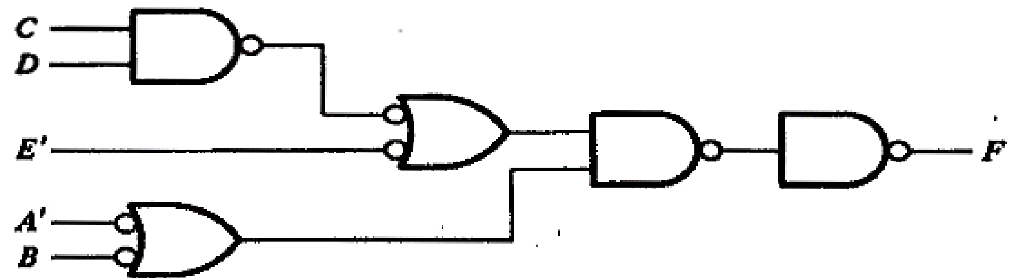
(γ) Διάγραμμα με πύλες ΟΧΙ-ΚΑΙ με ένα γραφικό σύμβολο .

Παράδειγμα (2)

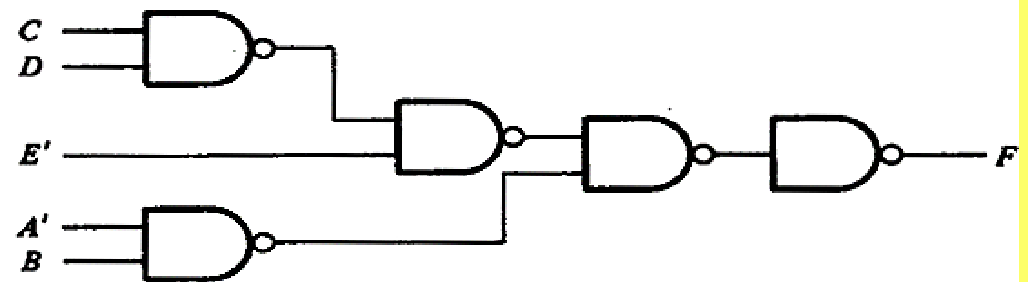
F=(CD+E)(A+B΄)



(α) Διάγραμμα ΚΑΙ-Ή



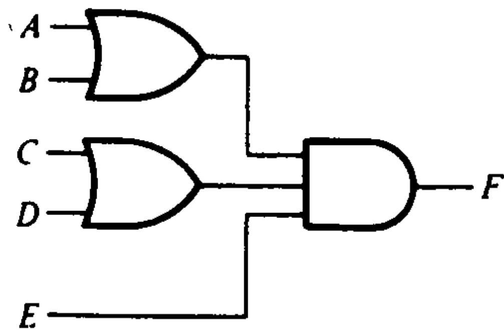
(β) Διάγραμμα ΟΧΙ-ΚΑΙ



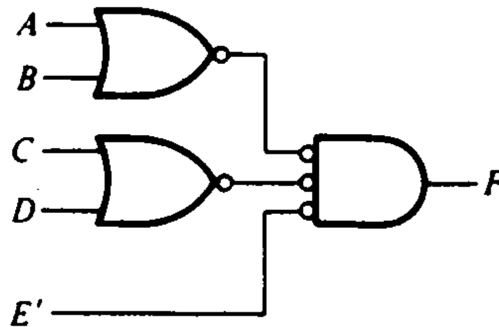
(γ) Εναλλακτικό διάγραμμα με πύλες ΟΧΙ-ΚΑΙ

Υλοποίηση με πύλες NOR

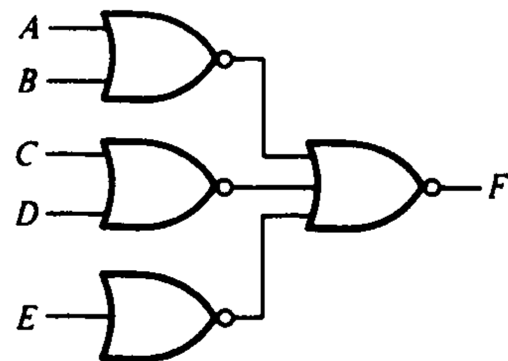
Η συνάρτηση NOR είναι το δυϊκό της NAND και άρα οι κανόνες μετατροπής είναι δυϊκοί.



(α) Ή-ΚΑΙ (OR-AND)



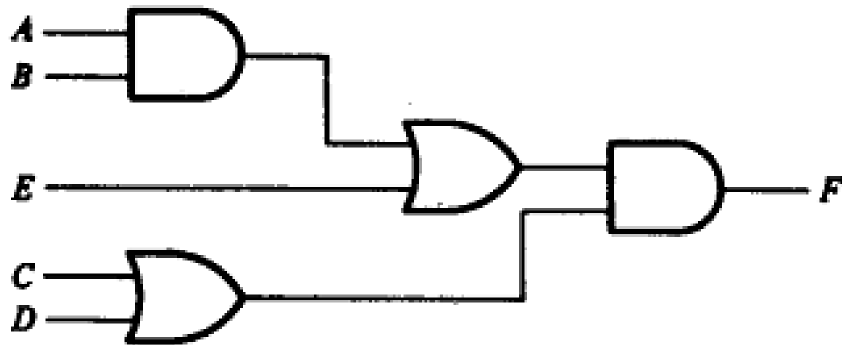
(β) ΟΥΤΕ-ΟΥΤΕ (NOR-NOR)



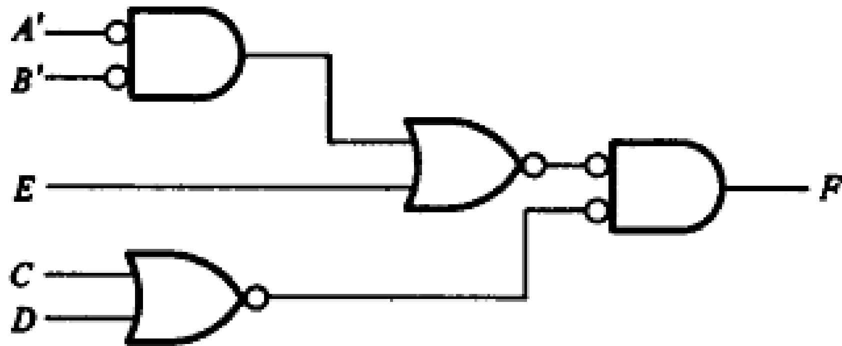
(γ) ΟΥΤΕ-ΟΥΤΕ (NOR-NOR)

Κυκλώματα NOR Πολλαπλών Επιπέδων

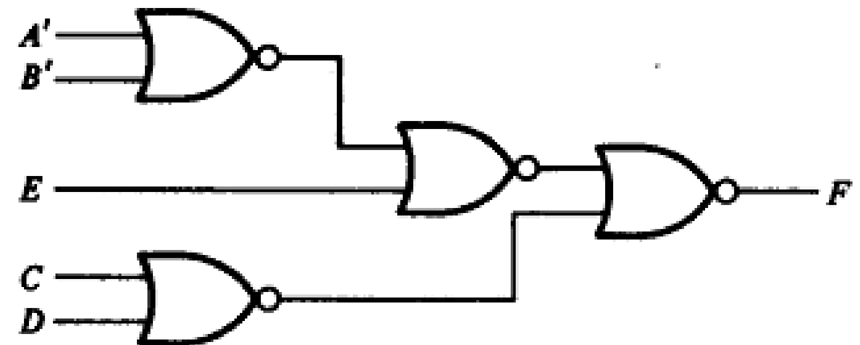
F = (AB + E)(C + D)



(α) Διάγραμμα ΚΑΙ-Ή



(β) Διάγραμμα με πύλες ΟΥΤΕ



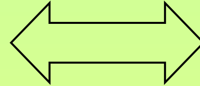
(γ) Εναλλακτικό διάγραμμα με πύλες ΟΥΤΕ

ΣΧΗΜΑ 4-19
Υλοποίηση της $F = (AB + E)(C + D)$ με πύλες ΟΥΤΕ

Η Συνάρτηση XOR

Αποκλειστικό Ή (XOR)
$x \oplus y = x'y + xy'$

Σχ. Αντιστρ.



Αποκλειστικό ΟΥΤΕ (XNOR)
$(x \oplus y)' = xy + x'y'$

- Ιδιότητες:

$x \oplus 0 = x$ $\qquad$ $x \oplus 1 = x'$

$x \oplus x = 0$ $\qquad$ $x \oplus x' = 1$

$x \oplus y' = (x \oplus y)'$ $\qquad$ $x' \oplus y = (x \oplus y)'$

- Η πράξη XOR είναι αντιμεταθετική και προσεταιριστική:

$$A \oplus B = B \oplus A$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C = A \oplus B \oplus C$$

- Δεν φτιάχνονται συχνά πύλες XOR με περισσότερες από 2 εισόδους.

Η Συνάρτηση XOR

Η συνάρτηση XOR πολλών μεταβλητών είναι περιττή: παίρνει τιμή 1 μόνο όταν περιττός αριθμός εισόδων είναι ίσος με 1.

A \ BC	00	01	11	10
0		1		1
1	1		1	

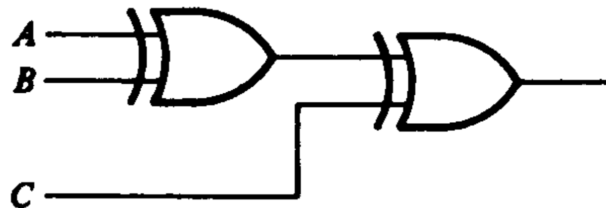
(α) Περιττή συνάρτηση

$F = A \oplus B \oplus C$

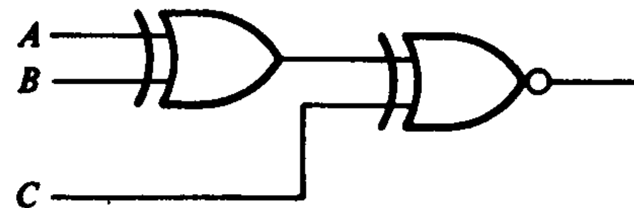
A \ BC	00	01	11	10
0	1		1	
1		1		1

(β) Άρτια συνάρτηση

$F = (A \oplus B \oplus C)'$



(α) Περιττή συνάρτηση τριών εισόδων



(β) Άρτια συνάρτηση τριών εισόδων

Η Συνάρτηση XOR

Μια συνάρτηση XOR *n* μεταβλητών είναι μια περιττή συνάρτηση που ορίζεται ως το λογικό άθροισμα των $2^n/2$ ελαχιστόρων των οποίων οι δυαδικές αριθμητικές τιμές τους έχουν περιττό αριθμό άσσων.

AB \ CD	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

(α) Περιττή συνάρτηση

$F = A \oplus B \oplus C \oplus D$

AB \ CD	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

(β) Άρτια συνάρτηση

$F = (A \oplus B \oplus C \oplus D)'$

Γεννήτρια και Ελεγκτής Ισοτιμίας

Πίνακας Αληθείας για τη Γεννήτρια Άρτιας Ισοτιμίας

Μήνυμα τριών bits			Bit Ισοτιμίας
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Ελεγκτής Άρτιας Ισοτιμίας

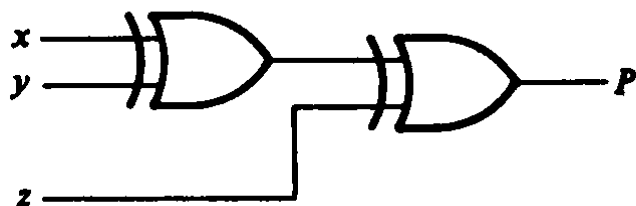
Τέσσερα Bits Δέκτη				Έλεγχος Λάθους Ισοτιμίας
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Αν ο αριθμός των 0 είναι άρτιος η συνάρτηση επαληθεύεται

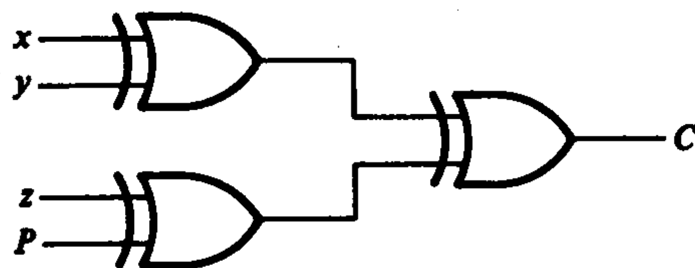
Άρα ο ελεγκτής λάθους θα επαληθεύεται αντίστροφα

Γεννήτρια και Ελεγκτής Ισοτιμίας

Τα κυκλώματα αυτά χρησιμοποιούνται στην ανίχνευση λαθών κατά τη μετάδοση ή λειτουργία των κυκλωμάτων.



(α) Γεννήτρια άρτιας ισοτιμίας τριών bits



(β) Ελεγκτής άρτιας ισοτιμίας τεσσάρων bits

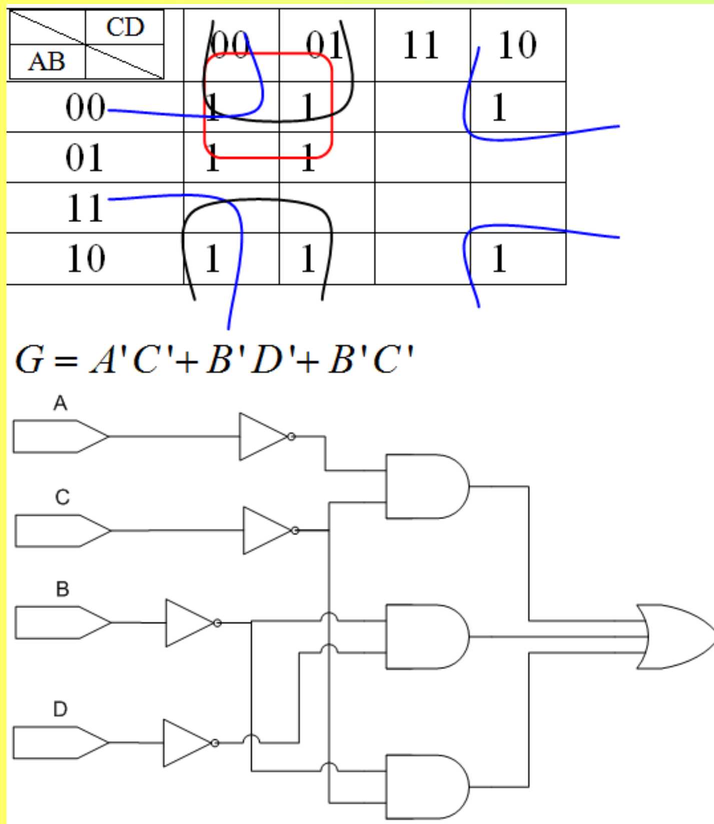
Το bit ισοτιμίας είναι περιττή πληροφορία η οποία όμως μπορεί να χρησιμοποιηθεί για την ανίχνευση μονού αριθμού λαθών.

Η μεγάλη εικόνα

- Κάθε μέθοδος απλοποίησης (Karnaugh που είδαμε και τις υπόλοιπες που θα δούμε πιο κάτω) μας δίνει ένα ελάχιστο (όχι απαραίτητα μοναδικό) κύκλωμα για υλοποίηση με πύλες NOT, AND και OR.
- Αν θέλω ελάχιστο αριθμό πυλών πρέπει να :
  - Κάνω αλγεβρικές απλοποιήσεις *επί της απλοποιημένης* μορφής της συνάρτησης ώστε να χρησιμοποιήσω πύλες NAND, NOR, XOR, XNOR.
  - Διαμοιράζομαι πύλες μεταξύ διαφόρων συναρτήσεων που σκοπεύω να υλοποιήσω ταυτόχρονα.

Αλγεβρικές απλοποιήσεις μετά το Karnaugh

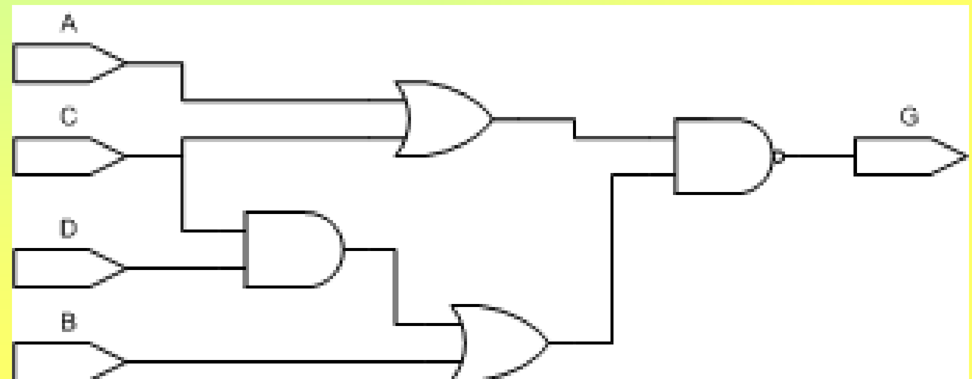
- G (A, B, C, D) = Σ(0, 1, 2, 4, 5, 8, 9, 10)
- Karnaugh & Λογικό διάγραμμα:



- Περαιτέρω αλγεβρική απλοποίηση
- G = (A+C)' + B'(C'+D') =

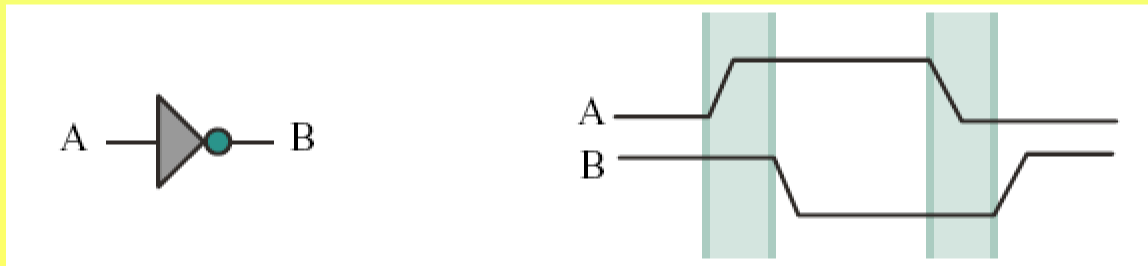
  (A+C)' + B'(CD)' =

  (A+C)' + (B+CD)' = ((A+C) (B+CD))'
- Νέο λογικό διάγραμμα :



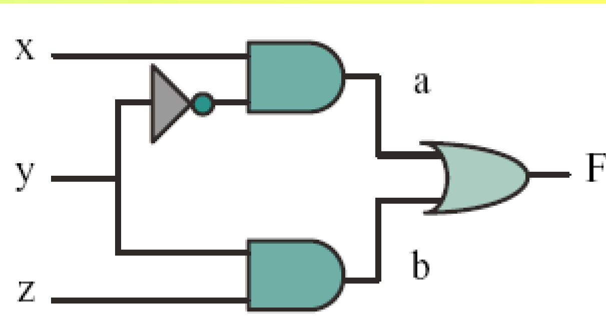
Απλοποίηση και πραγματικός κόσμος

- Μέχρι ώρας είδαμε την απλοποίηση κυκλωμάτων θεωρώντας μόνο τη λογική τους λειτουργία.
- Στο πραγματικό κόσμο, υπάρχει και μια άλλη διάσταση : ***η χρονική***.
- Θυμηθείτε ότι οι πύλες μοιάζουν με συστάδες διακοπτών, καθένας φτιαγμένος από τρανζίστορ.
- Κάθε τέτοιος διακόπτης χρειάζεται κάποιο χρόνο ώστε να αποκαταστήσει τη λειτουργία του, να μεταφέρει δηλαδή το σήμα από τη μία άκρη του στην άλλη.
- Ως εκ τούτου, κάθε πύλη έχει καθυστέρηση διάδοσης, οριζόμενη ως ο χρόνος που μεσολαβεί από την αλλαγή της εισόδου έως την αλλαγή της εξόδου που αυτή θα προκαλέσει.
- Η καθυστέρηση διάδοσης μπορεί να είναι διαφορετική για κάθε είδος μετάβασης της εξόδου. Αλλη δηλαδή η τιμή της για 0->1 και άλλη για 1->0.

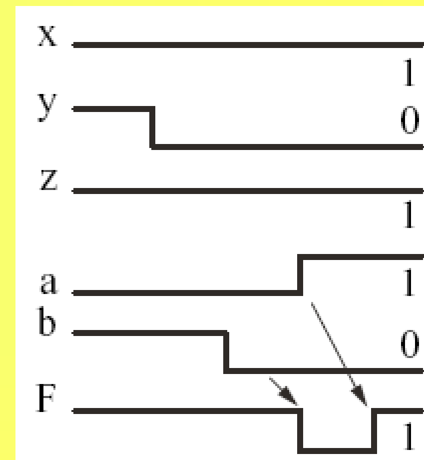


Στο πραγματικό κόσμο μπορεί να μη μας συμφέρει καν η απλοποίηση !

- Ας υποθέσουμε το κύκλωμα που ορίζεται από το παρακάτω πίνακα Karnaugh που μας οδηγεί στην απλοποίηση και την υλοποίηση :

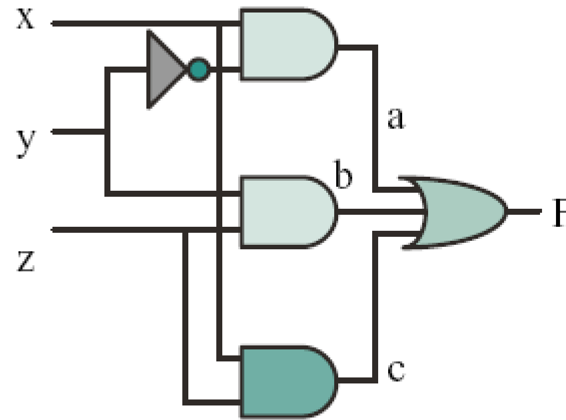


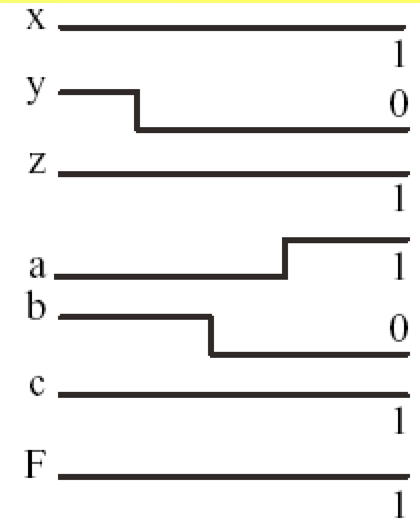
- **Αφού F(x, y, z) = x y' + yz, θα είναι F(1, y, 1) = y' + y = 1, δηλαδή η έξοδος δε θα έπρεπε να εξαρτάται από το y.**
- Στο πραγματικό κόσμο όμως η διαδρομή από την είσοδο y στο a είναι πιο αργή από την είσοδο y στο b λόγω της ύπαρξης του αντιστροφέα.
- Η έξοδος συνεπώς θα παρουσιάσει μια προσωρινή μηδενική τιμή για x=1, z=1 και y 1->0.
- Αυτή είναι μια ***αιχμή (static 1 hazard)***.
- Παρατηρείστε ότι αυτή συμβαίνει κατά τη μετάβαση από τον ελαχιστόρο m_7 στον m_3.
- Στατικές αιχμές της κατάστασης 1, εμφανίζονται κατά τις μεταβάσεις μεταξύ ελαχιστόρων που ανήκουν σε άλλες ομάδες



Λύση : μη απλοποιημένη συνάρτηση

x \ yz	00	01	11	10
0	0	2	6 1	4
1	1 1	3 1	7 1	5

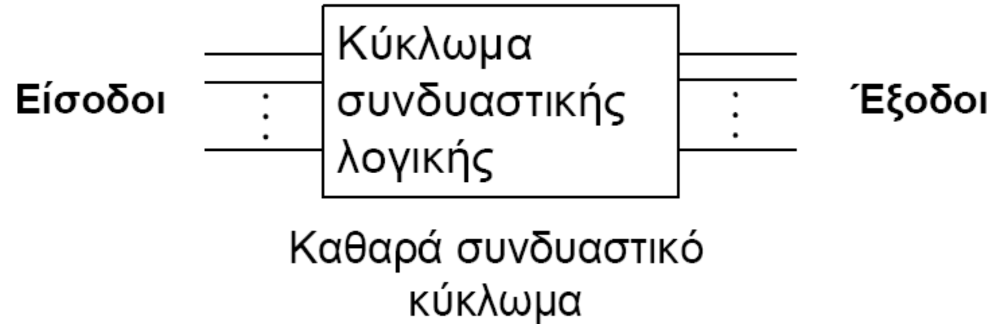




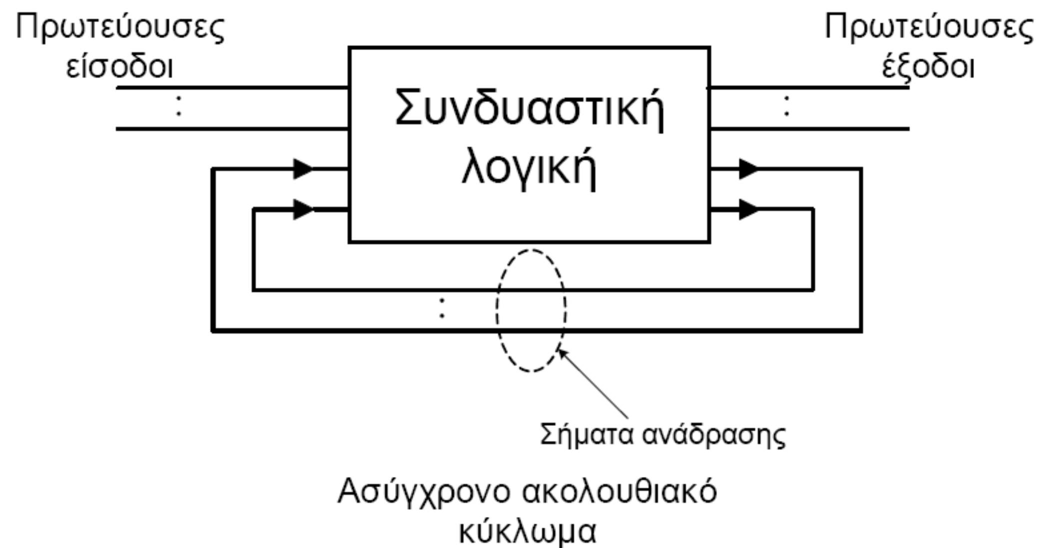
- Αν συμπεριλάβουμε το πρώτο συνεπαγωγό που καλύπτει τη μετάβαση από το m_3 στο m_7 το πρόβλημα της στατικής αιχμής στη κατάσταση 1 λύνεται.

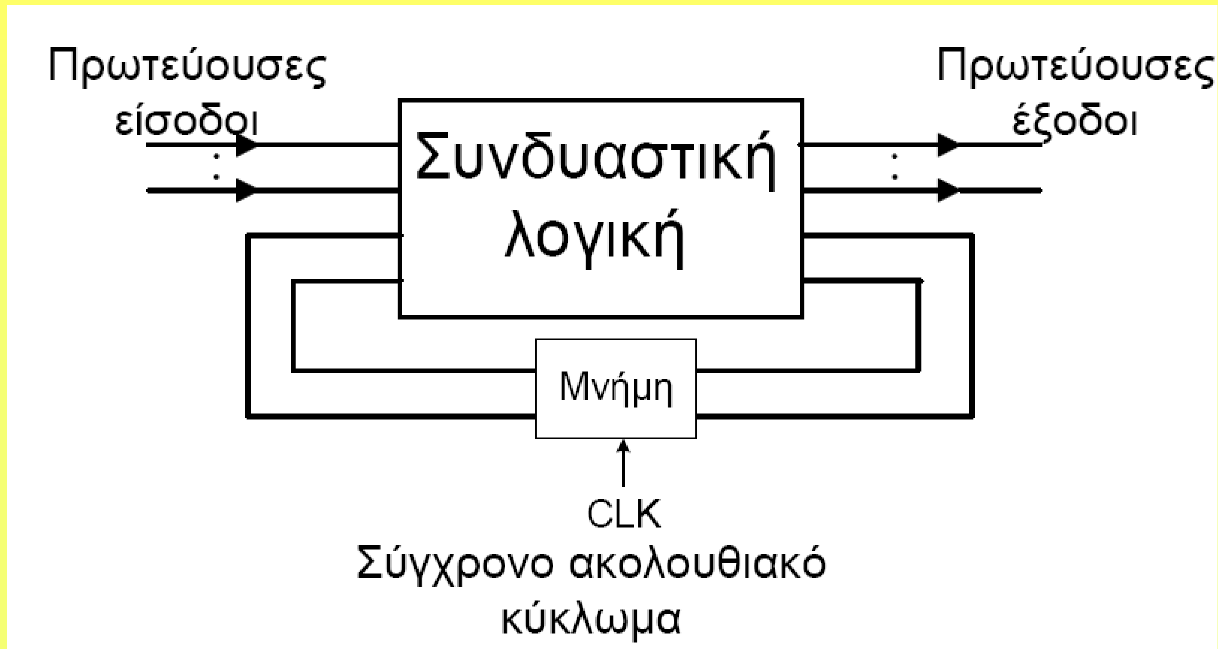
Γενικές κατηγορίες κυκλωμάτων

Συνδυαστικά



Ακολουθιακά





Τα **ακολουθιακά κυκλώματα** «θυμούνται» μέσω της σύνδεσης της ανάδρασης. Δύο είναι οι κύριες κατηγορίες των ακολουθιακών κυκλωμάτων:

Ασύγχρονα: Αλλάζουν κατάσταση σύμφωνα με τις αλλαγές των εισόδων τους. Απαιτούνται ειδικές τεχνικές σχεδιασμού.

Σύγχρονα: Τα σήματα ανάδρασης διακόπτονται από καταχωρητές που σκανδαλίζονται από παλμούς ρολογιού. Συνεπώς η κατάστασή του κυκλώματος αλλάζει σύμφωνα με τους παλμούς του ρολογιού. Η κατάσταση του κυκλώματος ορίζεται από το περιεχόμενο των στοιχείων της μνήμης.

Συνδυαστικά κυκλώματα που θα μας απασχολήσουν

Αριθμητικά κυκλώματα :

- Ημιαθροιστής
- Πλήρης αθροιστής
- Παράλληλος Αθροιστής / Αφαιρέτης
  - Με διάδοση κρατουμένου
  - Με πρόβλεψη κρατουμένου
- Συγκριτής
- Πολλαπλασιαστής

Κωδικοποίησης / Αποκωδικοποίησης / Πολυπλεξίας / Αποπλεξίας

Κωδικοποιητής

Κωδικοποιητής προτεραιότητας

Αποκωδικοποιητής / Αποπλέκτης

Πολυπλέκτης

Ημι-Αθροιστής (Half Adder)

1.<u>Καθορισμός προβλήματος</u>: κύκλωμα που να προσθέτει δύο δυαδικά ψηφία.

2.<u>Πλήθος εισόδων/εξόδων</u>: 2 είσοδοι – 2 έξοδοι.

3.<u>Ονομασία εισόδων/εξόδων</u>: έστω x, y οι δύο είσοδοι (προσθετέοι) και C (κρατούμενο), S (άθροισμα) οι δύο έξοδοι.

4.<u>Πίνακας αλήθειας</u>:

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Ημι-Αθροιστής

(α)

$S = x'y + xy'$

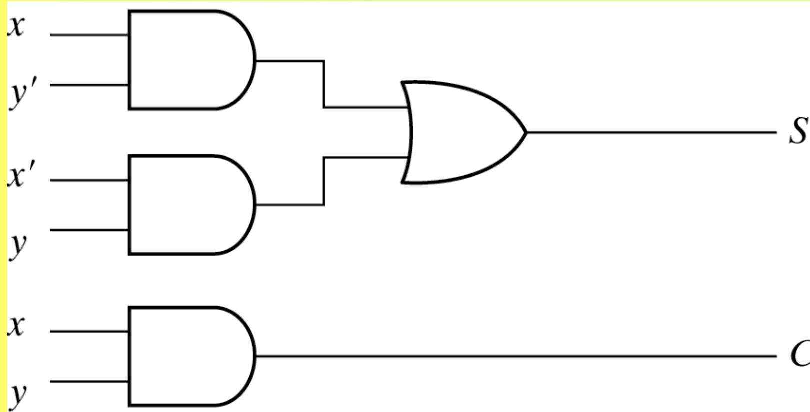
$C = xy$

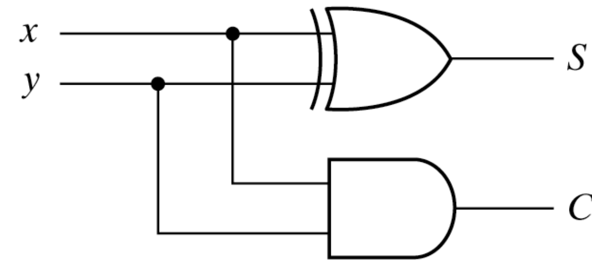
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(β)

$S = x \oplus y$

$C = xy$





Πλήρης Αθροιστής (Full Adder)

1.Καθορισμός προβλήματος: κύκλωμα που να προσθέτει τρία δυαδικά ψηφία.

2.Πλήθος εισόδων/εξόδων: 3 είσοδοι – 2 έξοδοι.

3.Ονομασία εισόδων/εξόδων: έστω x, y οι δύο είσοδοι (προσθετέοι), z το κρατούμενο της προηγούμενης βαθμίδας και C (κρατούμενο), S (άθροισμα) οι δύο έξοδοι.

4.Πίνακας αλήθειας:

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

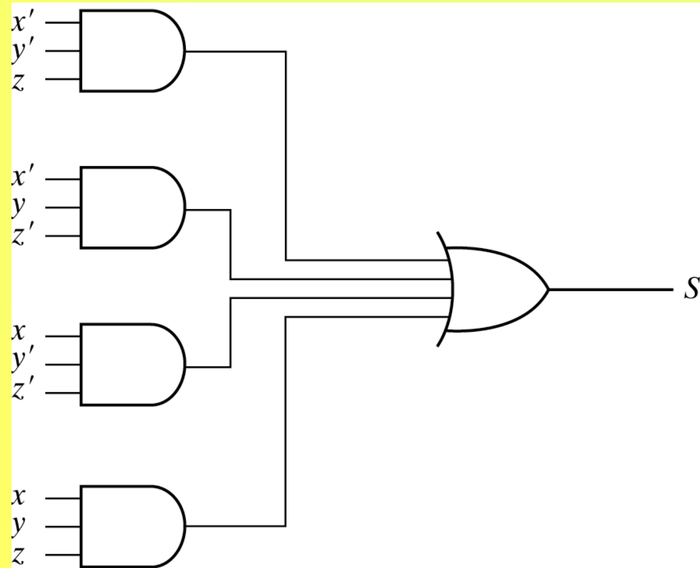
Πλήρης-Αθροιστής

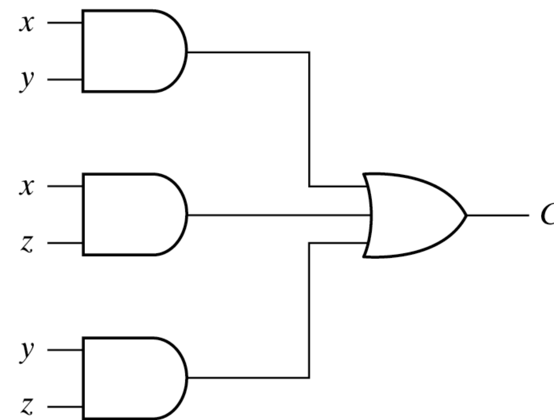
x \ yz	00	01	11	10
0		1		1
1	1		1	

x \ yz	00	01	11	10
0			1	
1		1	1	1

S = x′y′z+x′yz′+xy′z′+xyz

C = xy + yz + xz



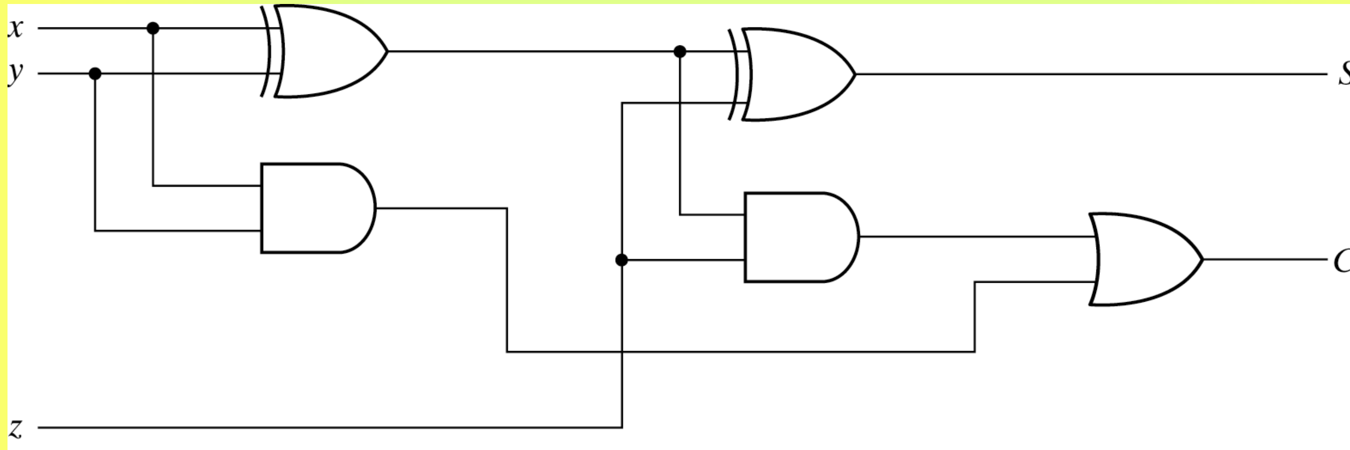


Πλήρης-Αθροιστής

- Είναι $S = (x \oplus y) \oplus z$
- Επίσης ισχύει

$$C = xy + yz + xz = xy + z\,(x + y) = xy + z\,(x'y + xy' + xy) =$$

$$xy + zxy + z\,(x'y + xy') = xy + z\,(x \oplus y)$$



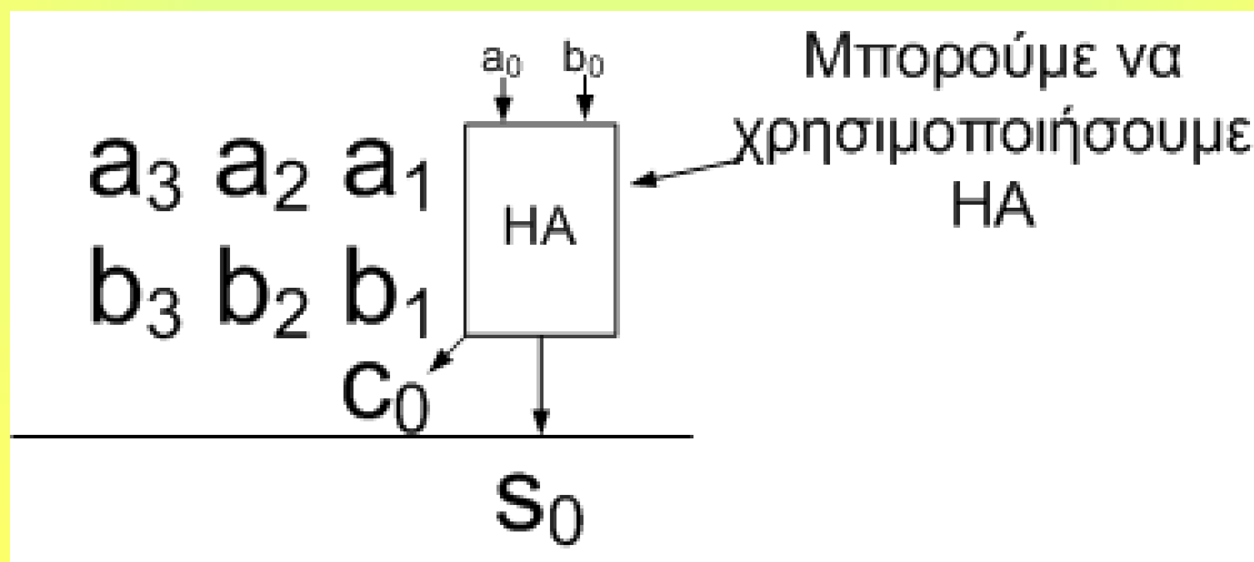
- Μπορώ συνεπώς να κατασκευάσω το πλήρη αθροιστή από 2 ημιαθροιστές και μια πύλη OR.

Πρόσθεση στο δυαδικό με διάδοση κρατουμένου (1/4)

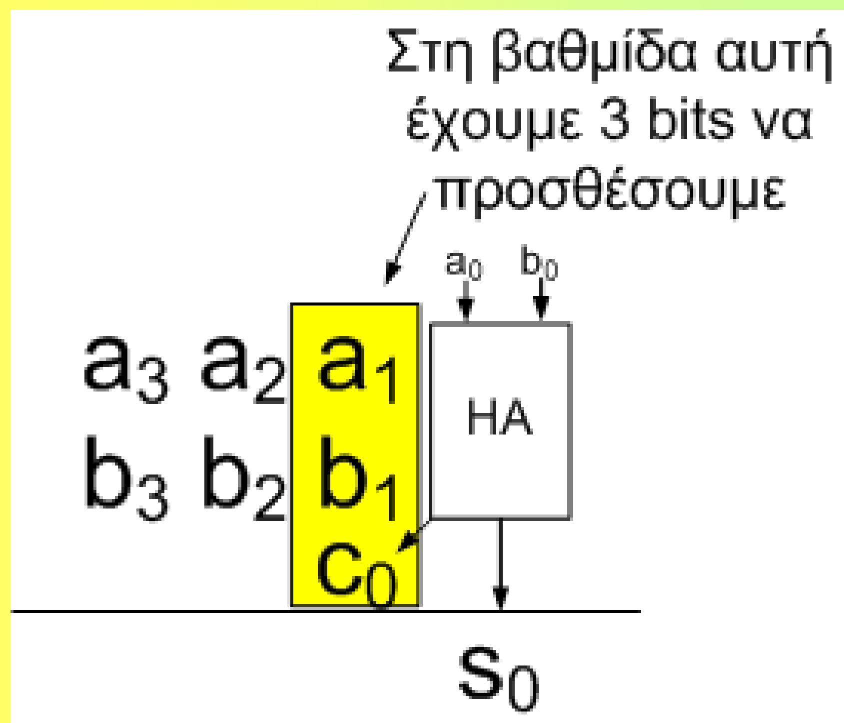
$$
\begin{array}{r}
a_3\ a_2\ a_1\ a_0 \\
b_3\ b_2\ b_1\ b_0\ + \\
\hline
s_3\ s_2\ s_1\ s_0
\end{array}
$$

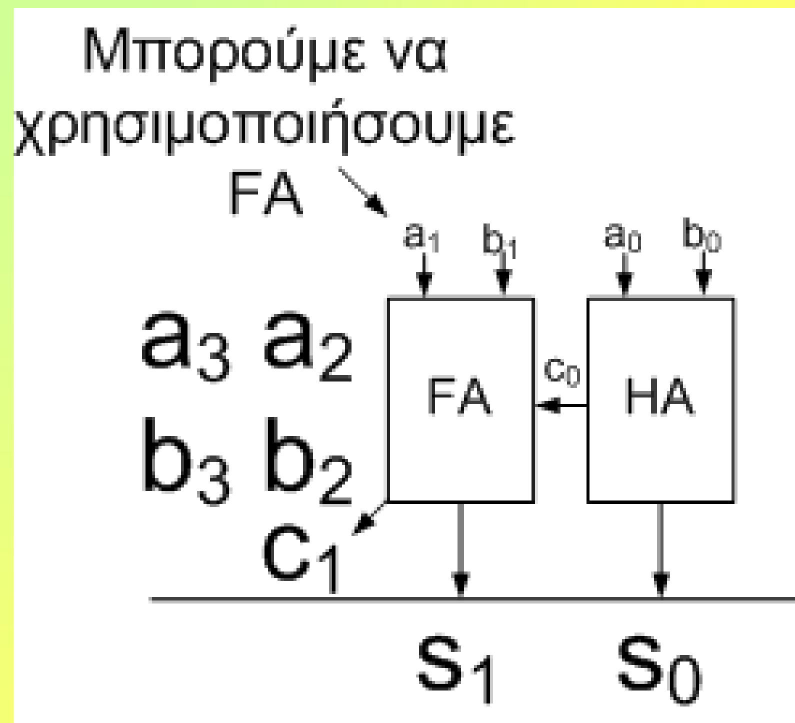
$$
\begin{array}{r}
a_3\ a_2\ a_1\ \boxed{a_0} \\
b_3\ b_2\ b_1\ \boxed{b_0}\ + \\
\hline
\end{array}
$$

Στη βαθμίδα αυτή έχουμε μόνο 2 bits να προσθέσουμε

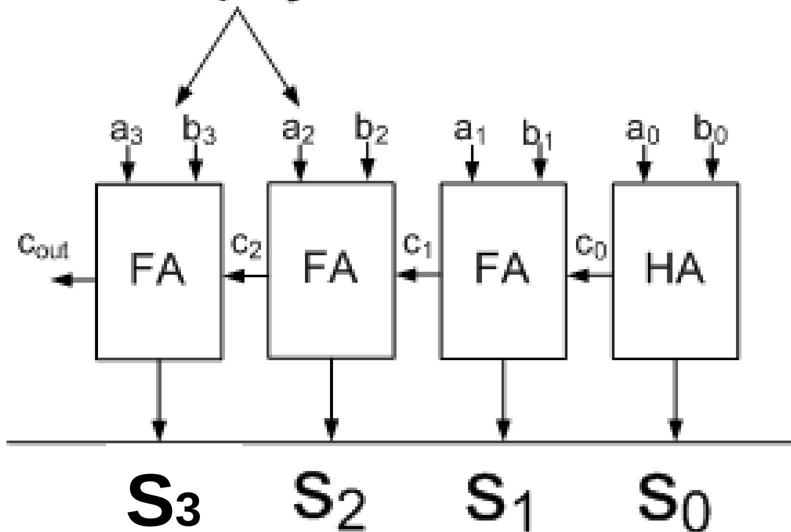


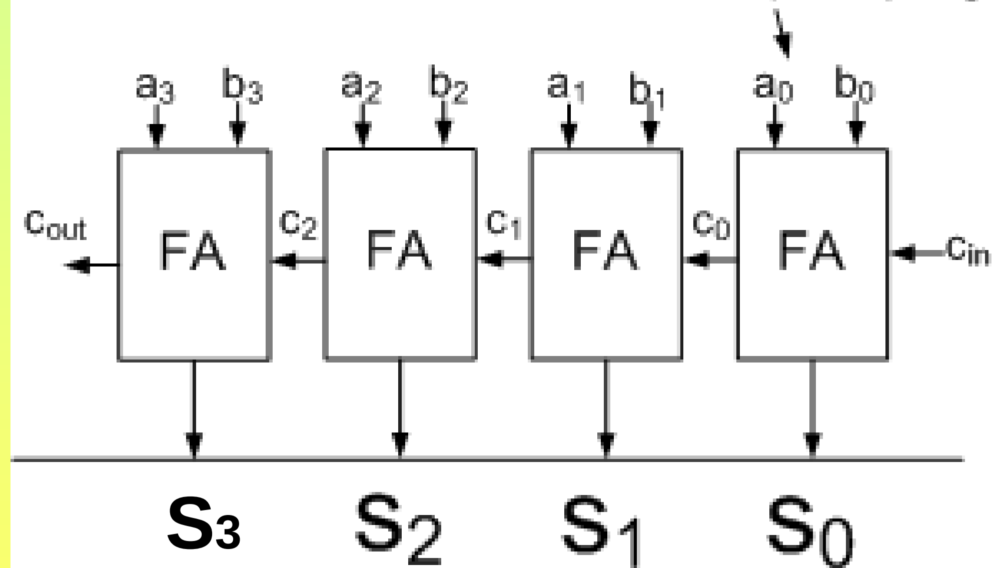
Πρόσθεση στο δυαδικό με διάδοση κρατουμένου (2/4)





Πρόσθεση στο δυαδικό με διάδοση κρατουμένου (3/4)



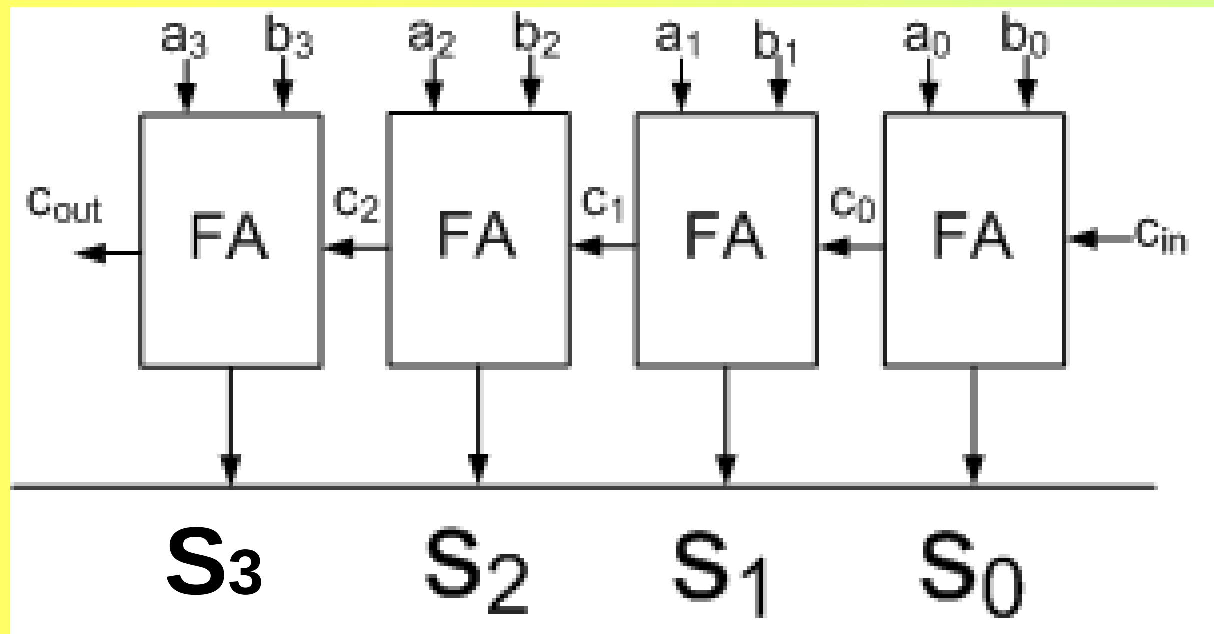


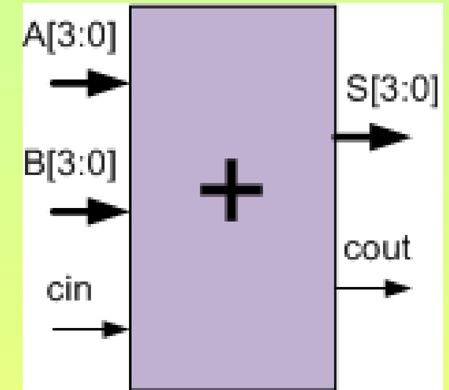
- Φτιάξαμε συνεπώς ένα κύκλωμα που μπορεί να προσθέτει έντελα των 4 δυαδικών ψηφίων.
- Το κύκλωμα αυτό θα ονομάζεται ***<u>παράλληλος αθροιστής</u>*** των 4 δυαδικών ψηφίων.
- Με την ίδια λογική μπορούμε να φτιάξουμε παράλληλους αθροιστές των κ δυαδικών ψηφίων
- Χρησιμοποιώντας δηλαδή MSI, φτιάχνουμε LSI (για κ ικανοποιητικά μεγάλο).

Πρόσθεση στο δυαδικό με διάδοση κρατουμένου (4/4)

- Στην ουσία στις προηγούμενες διαφάνειες δε περιγράψαμε ένα κύκλωμα, αλλά μια οικογένεια κυκλωμάτων.
- Όλα τα μέλη της οικογένειας ακολουθούν το ίδιο μοτίβο :
  - Αποτελούνται από πλήρεις αθροιστές.
  - Κάθε πλήρης αθροιστής προσθέτει ένα ζεύγος από αντίστοιχα δυαδικά ψηφία των εντέλων και το κρατούμενο της προηγούμενης βαθμίδας, παρέχει 1 δυαδικό ψηφίο του αποτελέσματος και κρατούμενο προς τον επόμενο αθροιστή.
- Κάθε τέτοιο καλώς οριζόμενο μοτίβο, στην επιστήμη μας είναι μια ***αρχιτεκτονική***.
- Περιγράψαμε συνεπώς παράλληλους αθροιστές με αρχιτεκτονική διάδοσης κρατουμένου (κάθε βαθμίδα διαδίδει τυχόν κρατούμενο στην επόμενη)

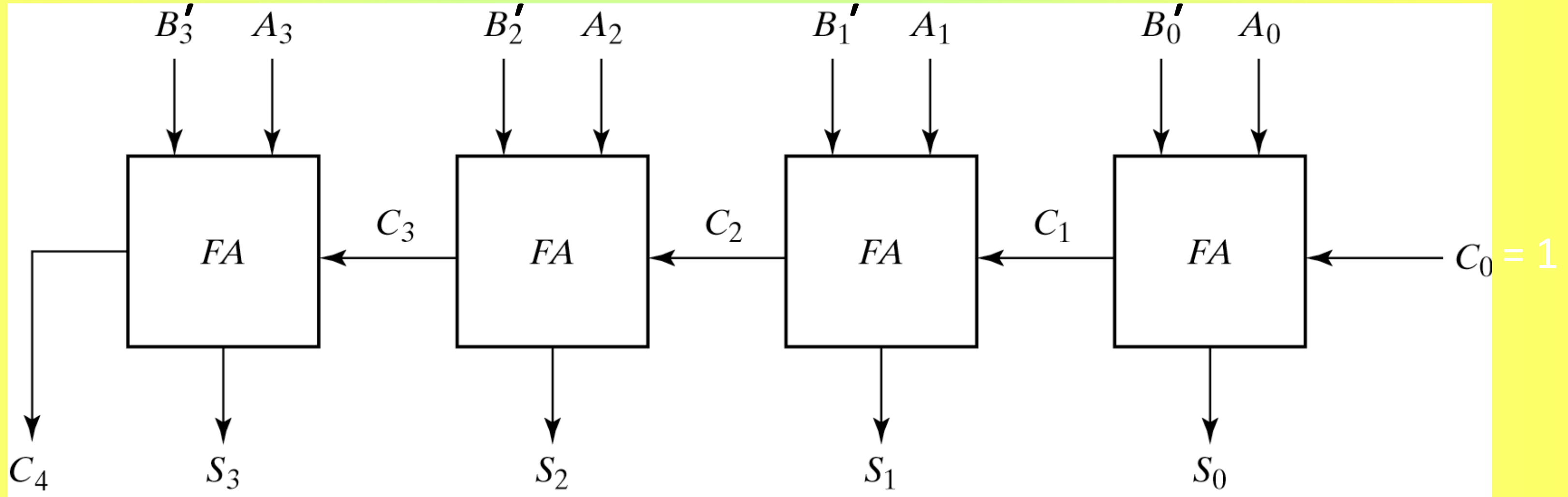
Παράλληλος δυαδικός αθροιστής με διάδοση κρατουμένου





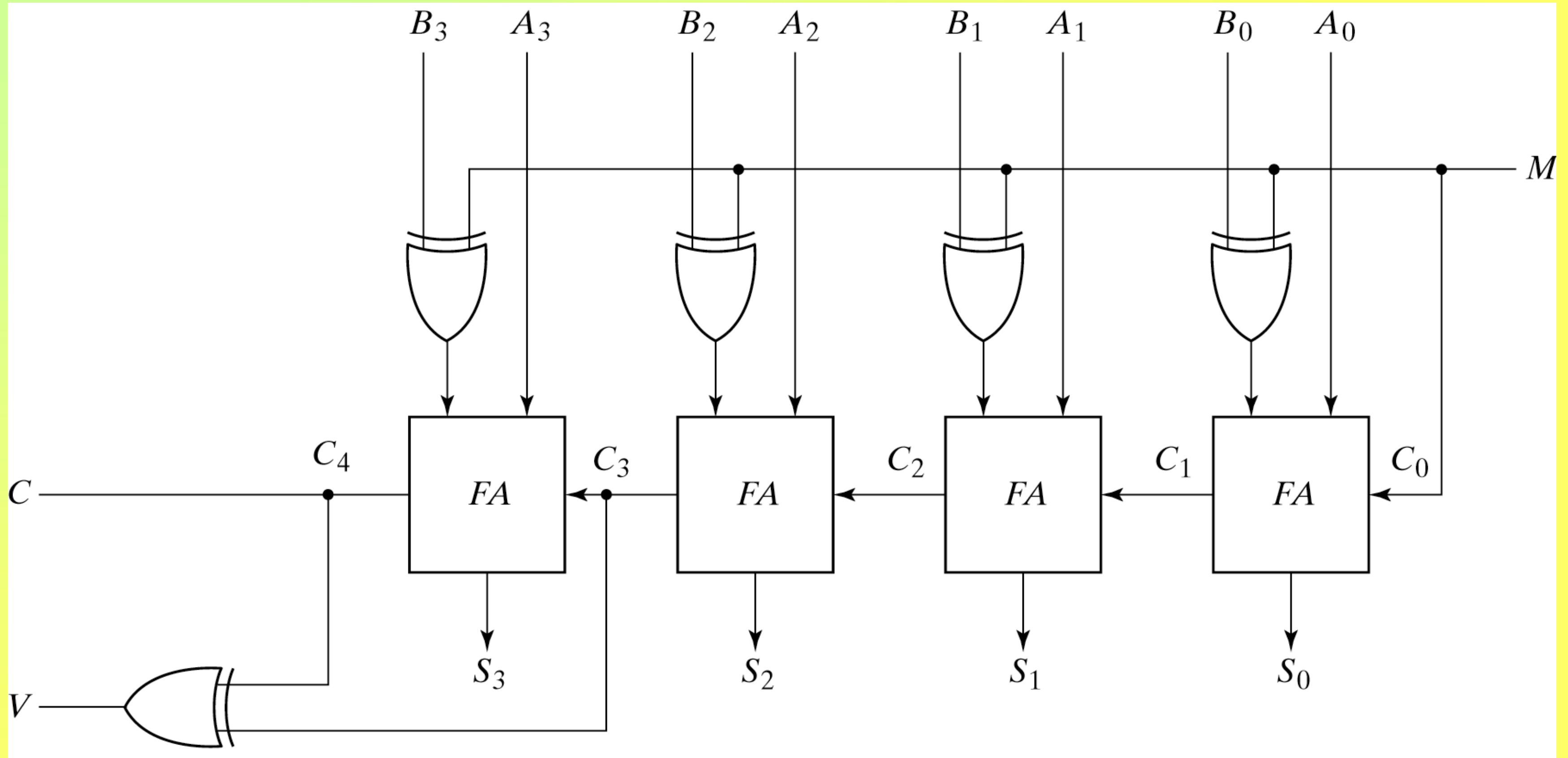
Υλοποίηση με συναρτήσεις: Πίνακας αληθείας με 9 εισόδους και 2^9=512 καταστάσεις.

Παράλληλος Δυαδικός Αφαιρέτης

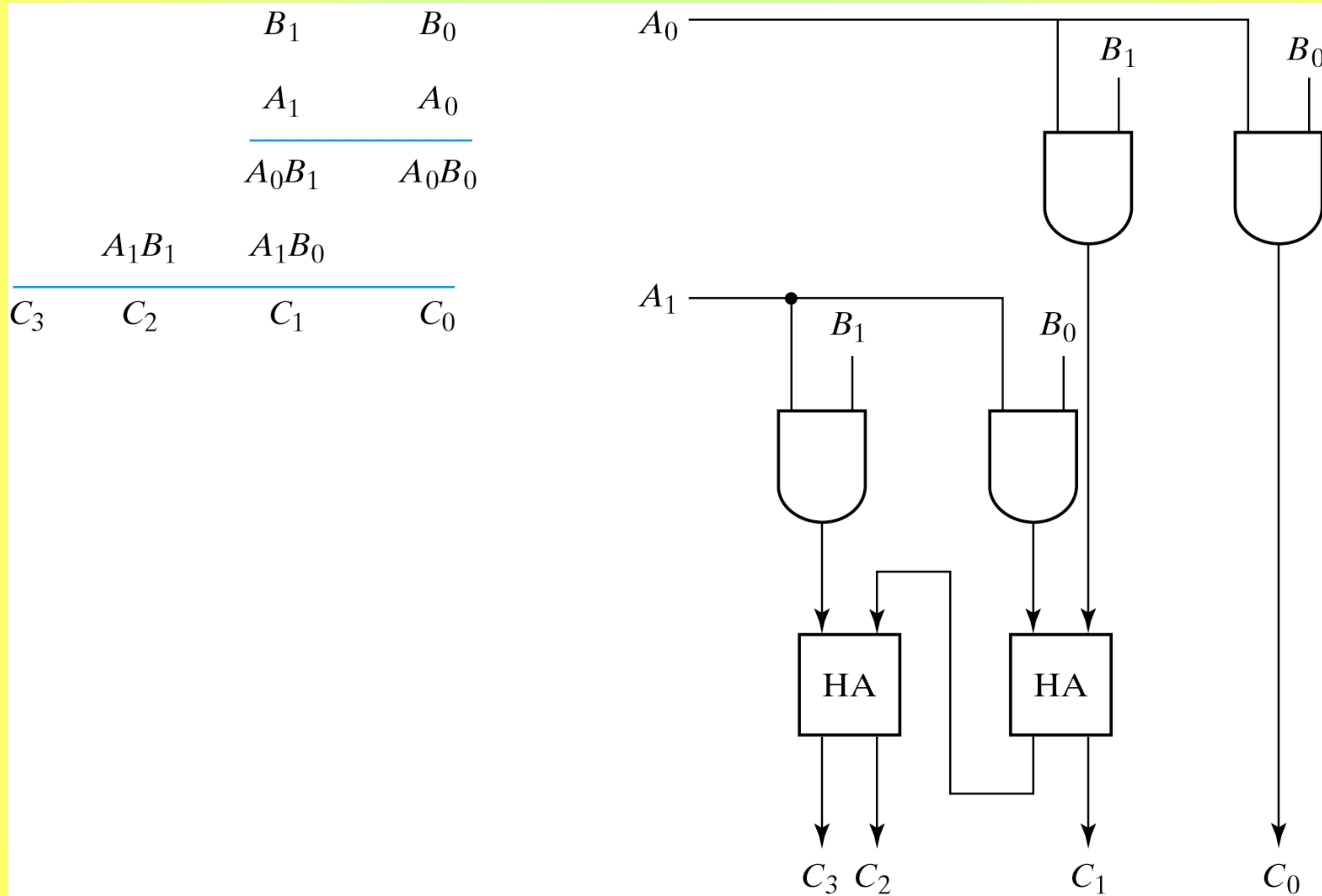


A – B = A + συμπλήρωμα(B) = A + B′ + 1

Παράλληλος Αθροιστής/Αφαιρέτης

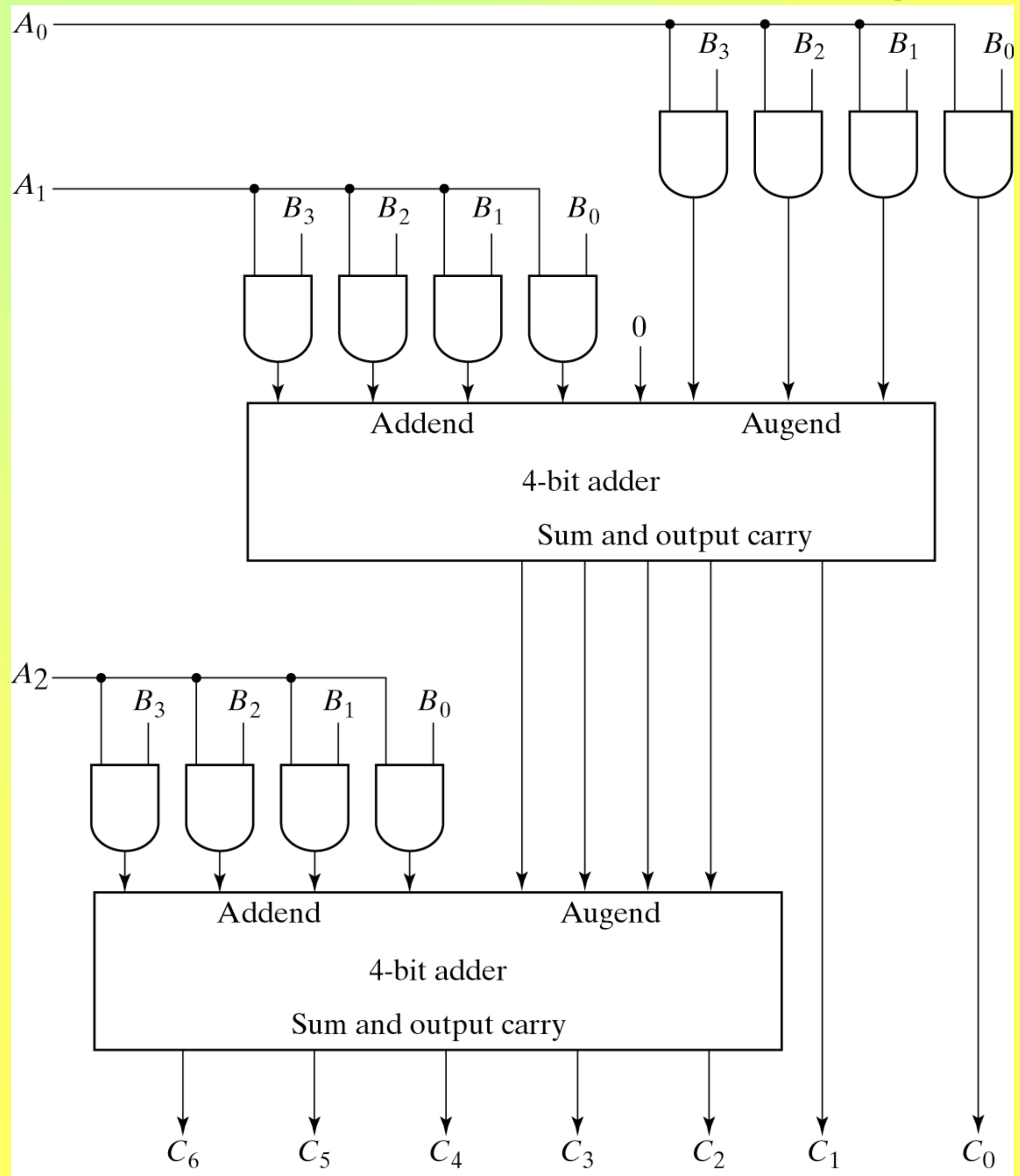


Δυαδικός πολλαπλασιαστής



Δυαδικός πολλαπλασιαστής

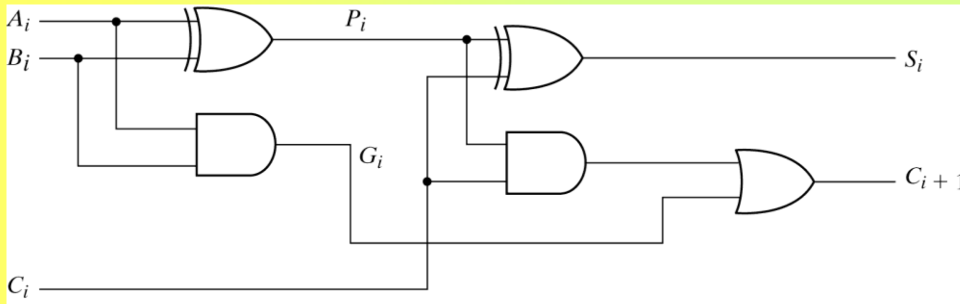
Κύκλωμα πολ/στη 4x3 bit



Διάδοση Κρατουμένου Δυαδικού Αθροιστή

Χρόνος Διάδοσης (Καθυστέρηση): Επίπεδα Πυλών × Καθυστέρηση Πύλης

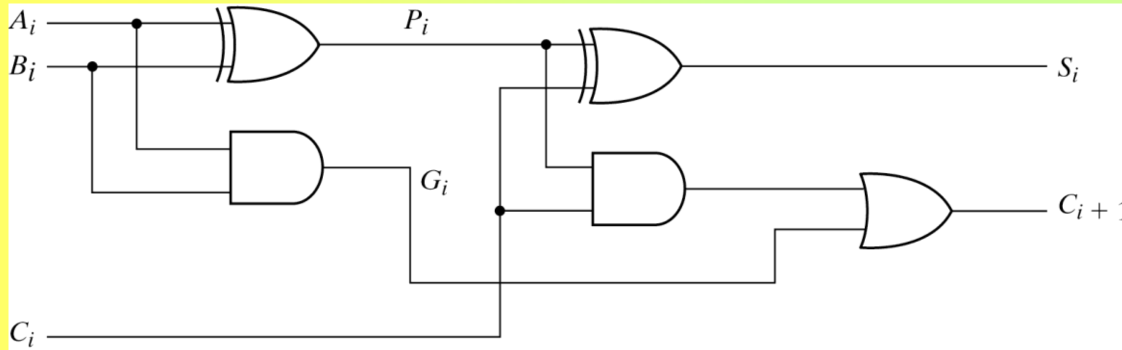
Παράλληλος Αθροιστής με διάδοση κρατουμένου : Η μεγαλύτερη καθυστέρηση οφείλεται στη διάδοση του κρατουμένου.



- 2 επίπεδα πυλών για κάθε διάδοση κρατουμένου.
- 2 × *n* επίπεδα για τον παράλληλο αθροιστή *n* bits.

- Η πρόσθεση είναι η πλέον συχνά χρησιμοποιούμενη πράξη.
- Η καθυστέρηση μπορεί να μειωθεί με τη χρήση γρηγορότερων πυλών. Ομως αυτή η λύση έχει σαφές άνω όριο οριζόμενο από την ισχύουσα τεχνολογία.
- Εναλλακτική λύση :
  - Η χρήση μιας πιο "παράλληλης" αρχιτεκτονικής.
  - Πιο ακριβή σε υλικό.

Πρόβλεψη Κρατουμένου (Carry Look-Ahead)



$P_i = A_i \oplus B_i$ $S_i = P_i \oplus C_i$

$G_i = A_i B_i$ $C_{i+1} = G_i + P_i C_i$

- Η σχέση $C_{i+1} = G_i + P_i C_i$ είναι αναδρομική
- Αναπτύσσοντας την αναδρομή, μπορούμε να πάρουμε τις εξής εξισώσεις :

C_0 = κρατούμενο εισόδου

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = \ldots = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

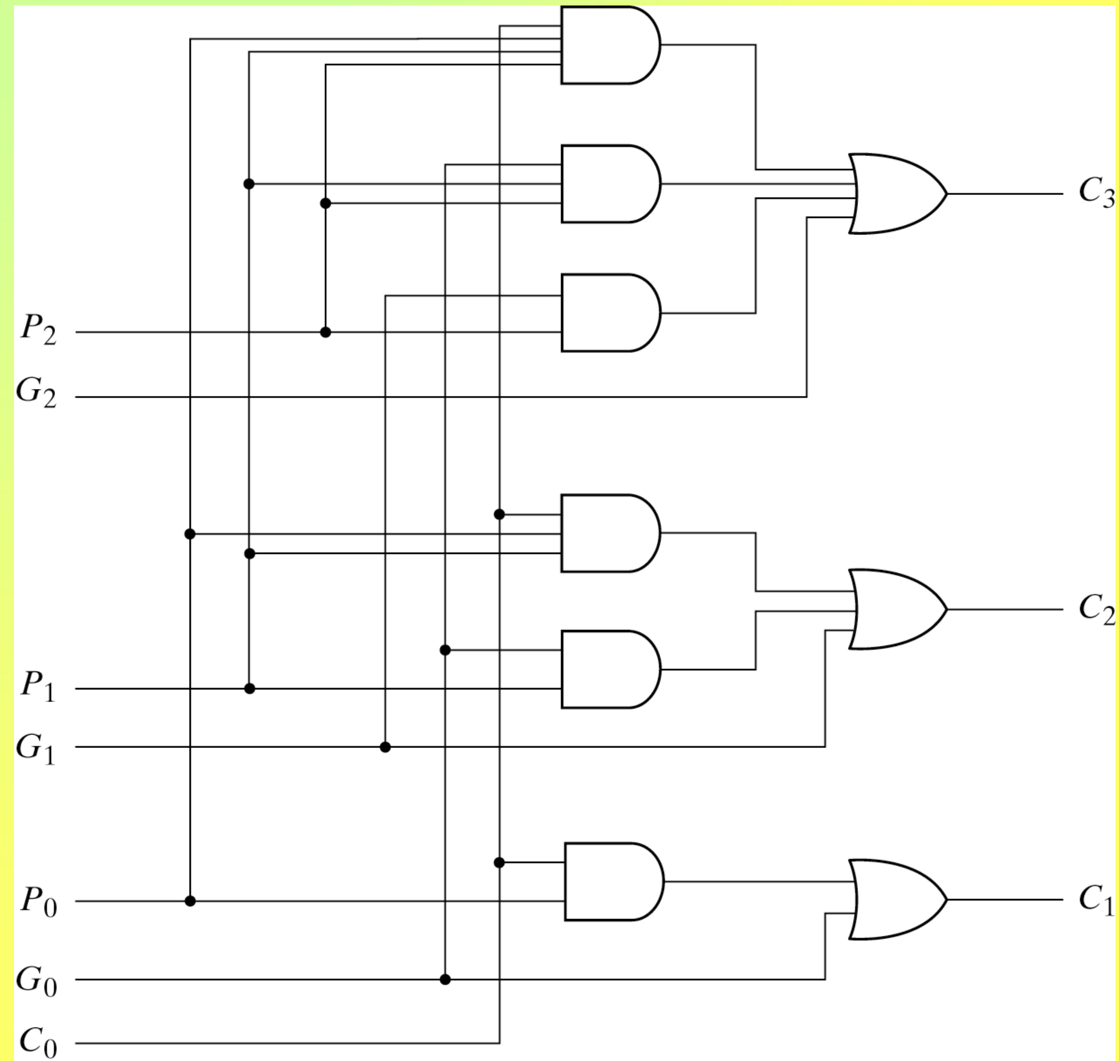
- Μπορούμε αυτές να τις υπολογίσουμε παράλληλα !

Γεννήτρια Πρόβλεψης Κρατουμένου

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

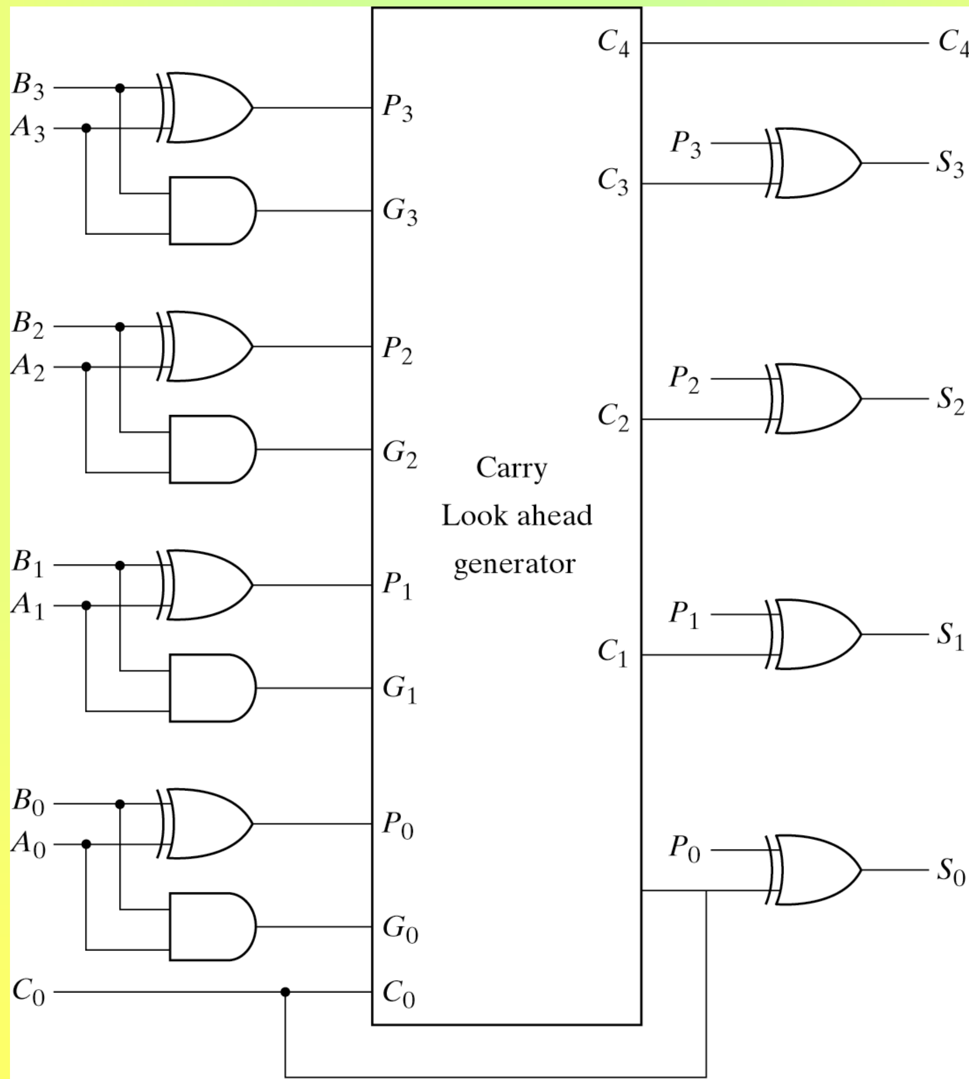
$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$



Αθροιστής Πρόβλεψης Κρατουμένου

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$



$S_i = P_i \oplus C_i$

Συγκριτής (Comparator)

<u>Συγκριτής Μεγέθους</u>: συγκρίνει δύο αριθμούς και βρίσκει τη σχέση τους (<,>,=).

Για δύο αριθμούς των *n* bits έχουμε 2^{2n} συνδυασμούς.

Το κύκλωμα του συγκριτή έχει αρκετή κανονικότητα.

- Έστω A = $A_3A_2A_1A_0$ και B = $B_3B_2B_1B_0$ οι δύο αριθμοί.

- Ισχύει A = B όταν όλα τα ζευγάρια (A_i, B_i) είναι ίσα, δηλαδή A_3=B_3 και A_2=B_2 και A_1=B_1 και A_0=B_0.

$$(A=B) = x_3x_2x_1x_0 \qquad x_i = A_iB_i + A_i'B_i'$$

Συγκριτής Μεγέθους

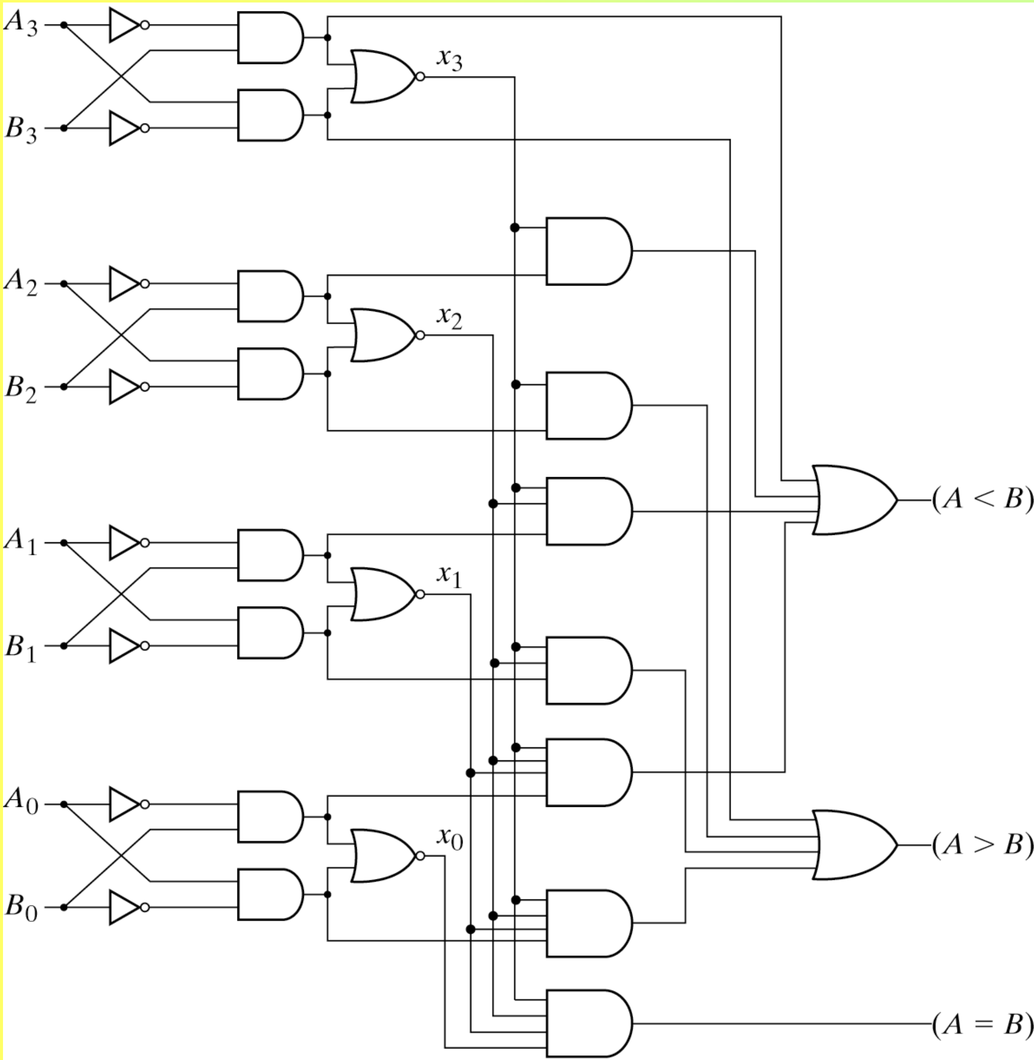
- Για να βρούμε εάν $A<B$ ή $A>B$ εξετάζουμε τα σχετικά μεγέθη των ζευγαριών ψηφίων ξεκινώντας από την πιο σημαντική θέση. Εάν τα δύο ψηφία είναι ίσα τότε συγκρίνουμε το επόμενο λιγότερο σημαντικό ζευγάρι ψηφίων.

- Εάν $A_i = 1$ και $B_i = 0$ τότε $A > B$, εάν $A_i = 0$ και $B_i = 1$ τότε $A < B$.

$$(A>B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A<B) = B_3A_3' + x_3B_2A_2' + x_3x_2B_1A_1' + x_3x_2x_1B_0A_0'$$

Συγκριτής Μεγέθους



$(A=B) = x_3x_2x_1x_0$ $\quad x_i = A_iB_i + A_i'B_i'$

$(A>B)=A_3B_3'+x_3A_2B_2'+x_3x_2A_1B_1'+x_3x_2x_1A_0B_0'$

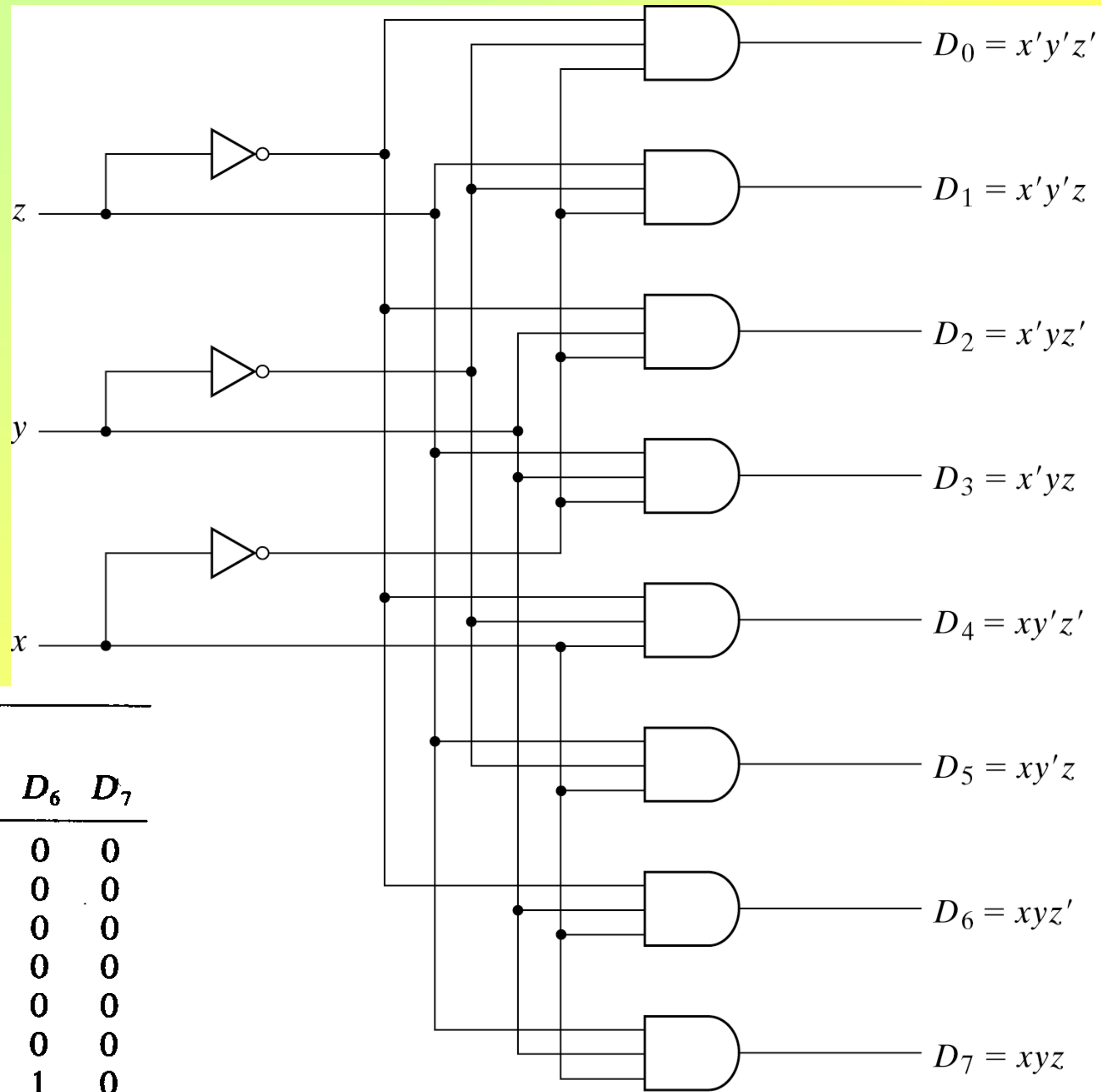
$(A<B)=B_3A_3'+x_3B_2A_2'+x_3x_2B_1A_1'+x_3x_2x_1B_0A_0'$

Αποκωδικοποιητής (Decoder)

<u>Αποκωδικοποιητής</u>: κύκλωμα που μετατρέπει τη δυαδική πληροφορία των *n* γραμμών εισόδου σε εως 2^n μοναδικές γραμμές εξόδου (ελαχιστόροι *n* μεταβλητών).

<u>Παράδειγμα</u>: Αποκωδικοποιητής 3-σε-8

Ο αποκωδικοποιητής παράγει τον 1 από 2^n κώδικα.

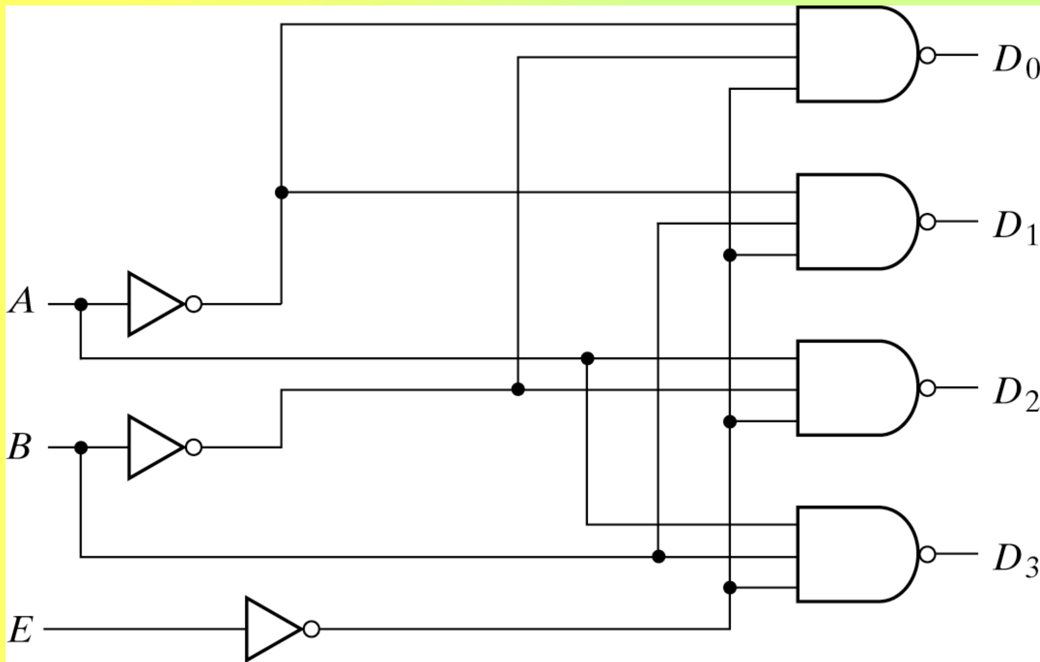


Είσοδοι			Έξοδοι							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Αποκωδικοποιητής με Είσοδο Επίτρεψης

Ο αποκωδικοποιητής μπορεί να παράγει συμπληρωματικές εξόδους.

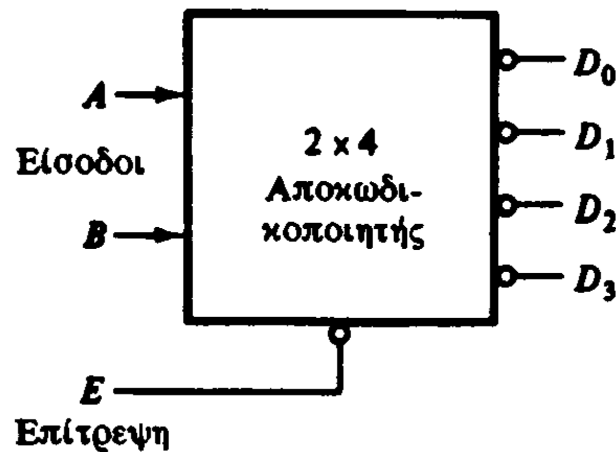
Ο αποκωδικοποιητής μπορεί να έχει είσοδο επίτρεψης. (Σε αυτή τη περίπτωση συνήθως χρησιμοποιούμε την ορολογία αποπλέκτης).



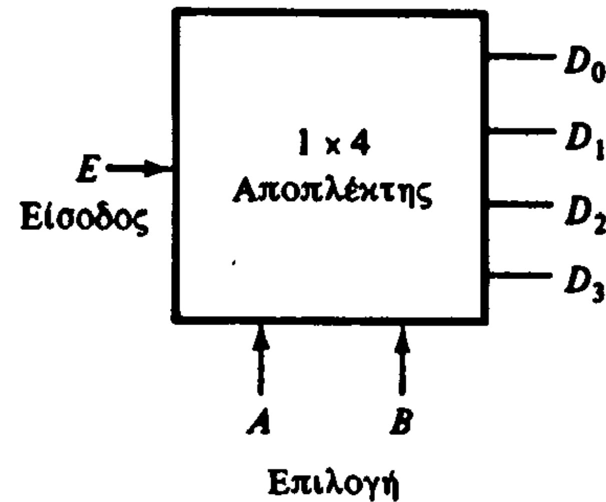
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Λειτουργία ως αποπλέκτης (Demultiplexer)

Ο αποπλέκτης δέχεται πληροφορίες από μία απλή γραμμή και τις μεταβιβάζει σε μία από τις 2^n δυνατές γραμμές εξόδου ανάλογα με τις τιμές των *n* γραμμών επιλογής.



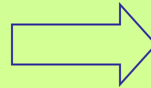
(α) Αποκωδικοποιητής με επίτρεψη



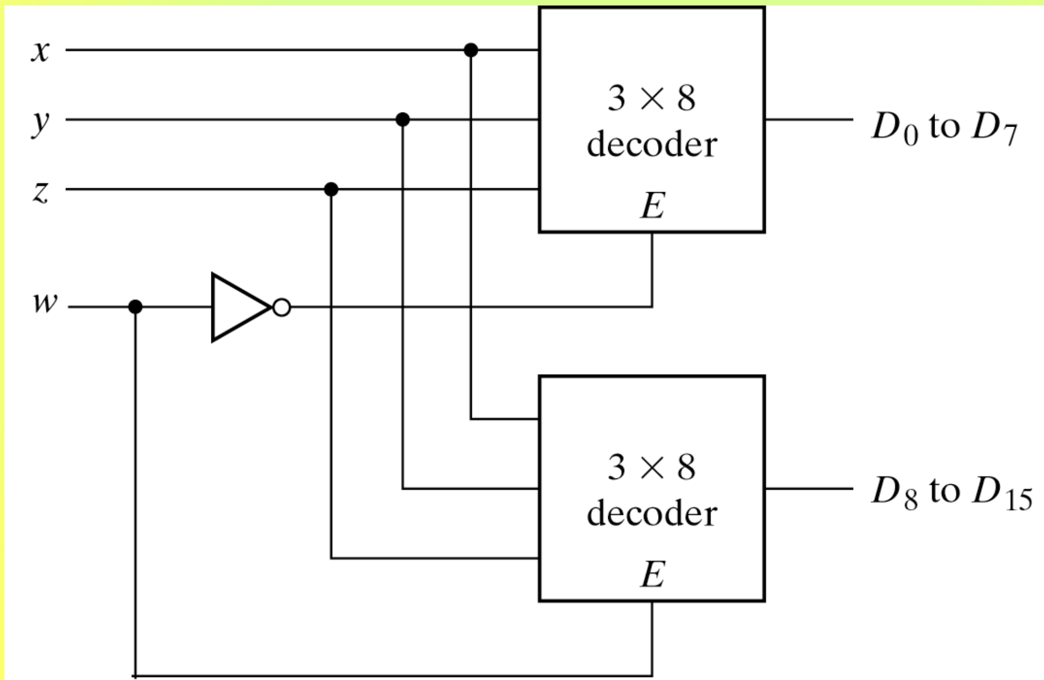
(β) Αποπλέκτης

Αποκωδικοποιητής/Αποπλέκτης

Επέκταση αποκωδικοποιητή με χρήση πολλών αποπλεκτών



2 αποπλέκτες 3 σε 8 δίνουν
1 αποκωδικοποιητή 4 σε 16



Υλοποίηση Συνάρτησης με Αποκωδικοποιητή

- Αφού ο αποκωδικοποιητής παράγει τους 2^n ελαχιστόρους μπορεί να χρησιμοποιηθεί για να υλοποιήσει οποιαδήποτε συνάρτηση.

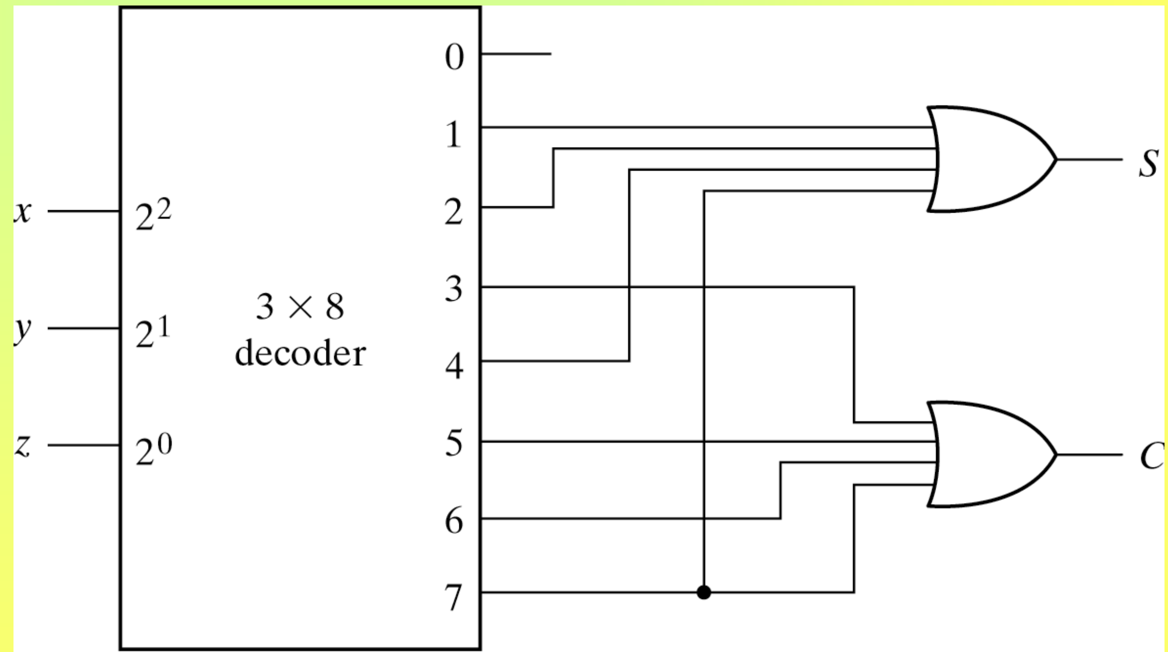
- Κάθε συνδυαστικό κύκλωμα με *n* εισόδους και *m* εξόδους μπορεί να υλοποιηθεί με έναν αποκωδικοποιητή *n*-σε-2^n γραμμών και *m* πύλες OR.

- Εάν ο αριθμός των ελαχιστόρων μιας συνάρτησης είναι μεγαλύτερος από $2^n/2$, τότε μπορούμε να χρησιμοποιήσουμε μία πύλη NOR για να αθροίσουμε τους ελαχιστόρους της F΄. Η έξοδος της πύλης NOR δίνει τη συνάρτηση F.

Παράδειγμα

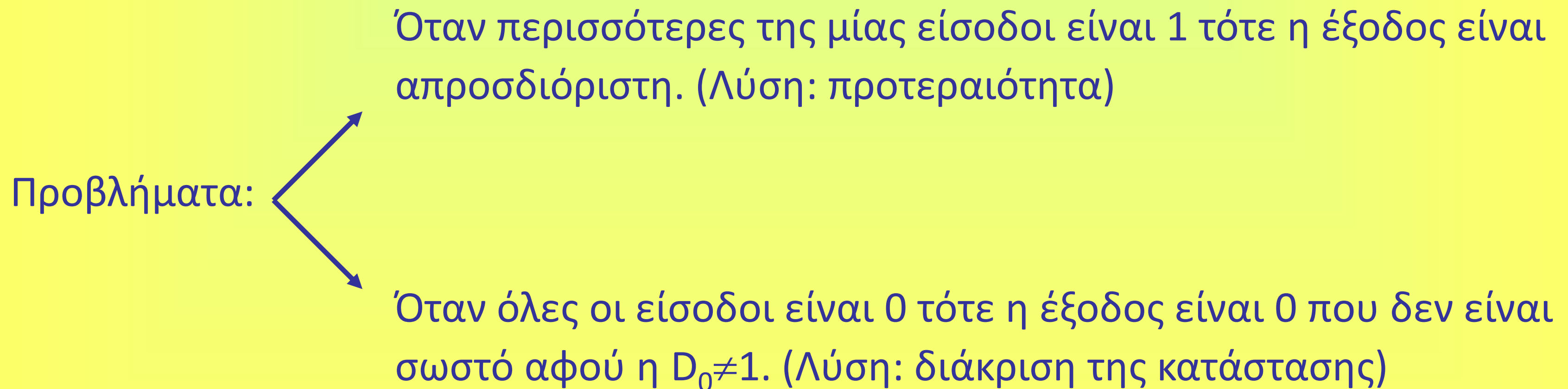
Υλοποίηση πλήρους αθροιστή με αποκωδικοποιητή.

S(x,y,z) = Σ(1,2,4,7)
C(x,y,z) = Σ(3,5,6,7)



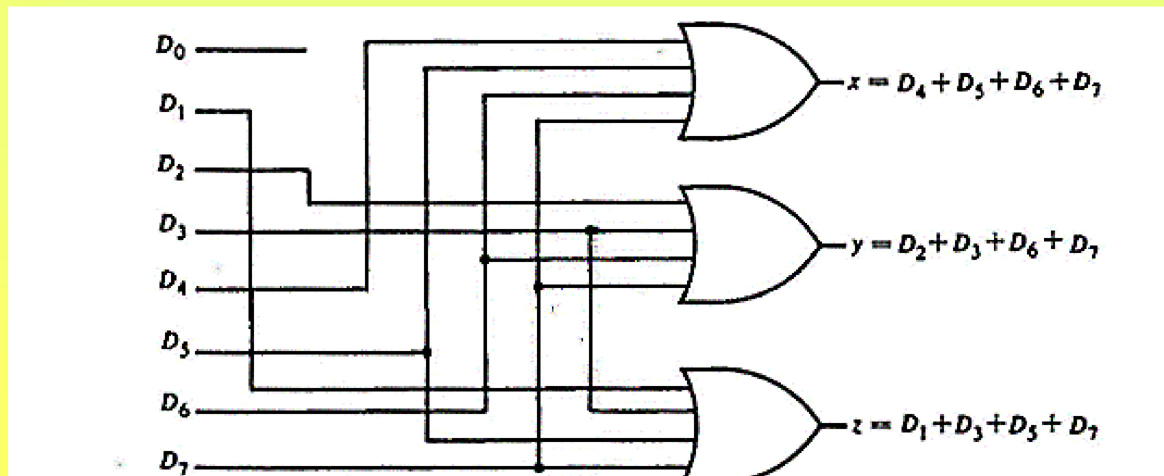
Κωδικοποιητής (Encoder)

Ο Κωδικοποιητής εκτελεί την αντίστροφη λειτουργία από τον Αποκωδικοποιητή: Έχει 2^n γραμμές εισόδου και *n* γραμμές εξόδου και δίνει στην έξοδο τον δυαδικό κώδικα που αντιστοιχεί στις γραμμές εισόδου.



Κωδικοποιητής από Οκτώ σε 3 δυαδικά ψηφία

Είσοδοι								Έξοδοι		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



Πολυπλέκτης (Multiplexer)

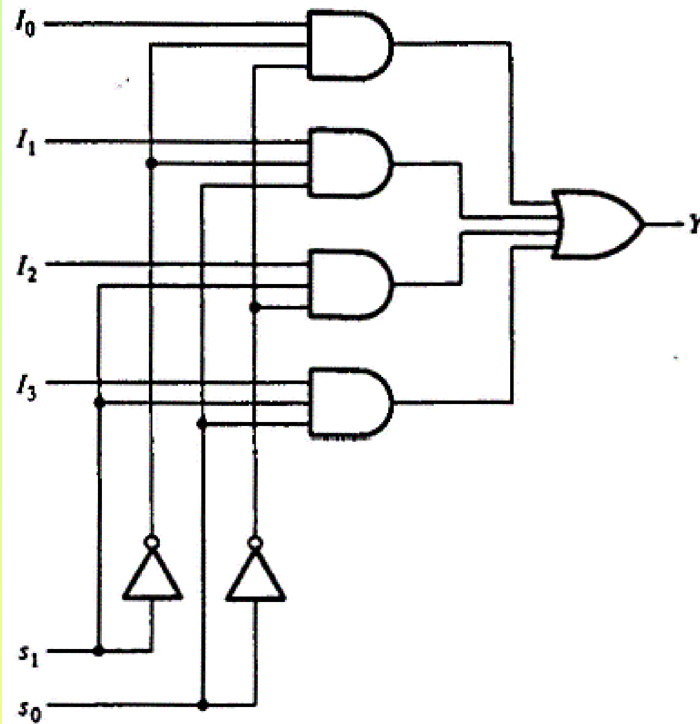
- Ο πολυπλέκτης είναι ένα συνδυαστικό κύκλωμα που επιλέγει δυαδικές πληροφορίες ανάμεσα σε πολλές γραμμές εισόδου και τις κατευθύνει σε μία γραμμή εξόδου.

- Η επιλογή της μιας συγκεκριμένης γραμμής εισόδου γίνεται μέσω μερικών γραμμών επιλογής.

- Ένας πολυπλέκτης 2^n-σε-1 γραμμή κατασκευάζεται από έναν αποκωδικοποιητή n-σε-2^n προσθέτοντας σε αυτόν 2^n εισόδους μια για κάθε πύλη AND. Οι έξοδοι των πυλών AND εφαρμόζονται σε μια μοναδική πύλη OR για να δώσουν τη μία γραμμή εξόδου.

Παράδειγμα

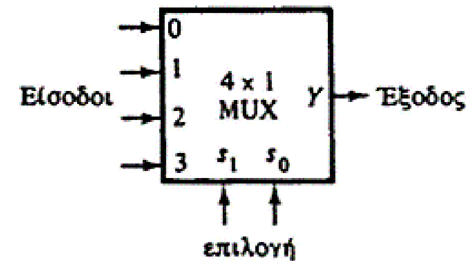
Πολυπλέκτης 4-σε-1



(α) Λογικό διάγραμμα

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

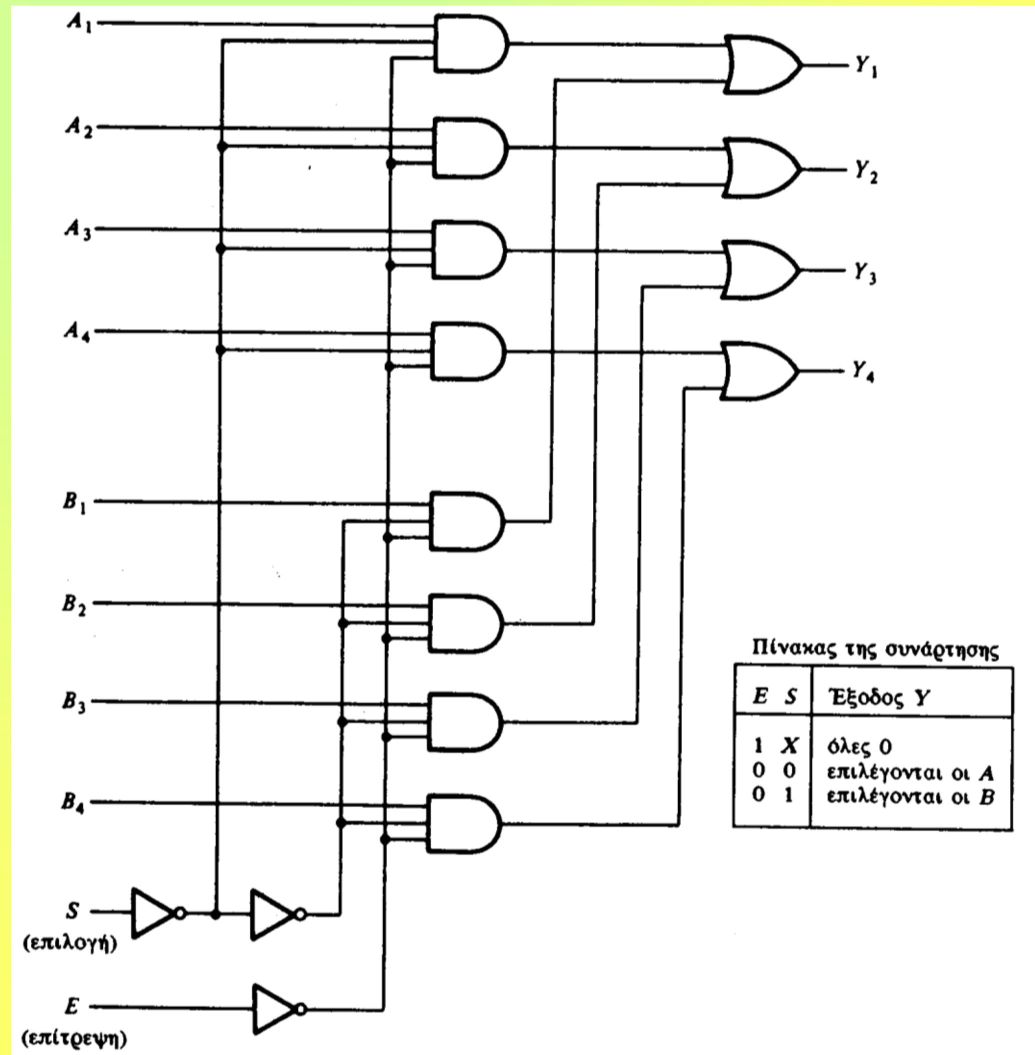
(β) Πίνακας της συνάρτησης



(γ) Σχηματικό διάγραμμα

Πολυπλέκτες με Κοινή Είσοδο Επίτρεψης

Η είσοδος Επίτρεψης (Ε) τοποθετείται για λόγους επέκτασης.



E	S	Έξοδος Y
1	X	όλες 0
0	0	επιλέγονται οι A
0	1	επιλέγονται οι B

Υλοποίηση Συνάρτησης με Πολυπλέκτη

Κάθε πολυπλέκτης 2^n σε 1 μπορεί να υλοποιήσει οποιαδήποτε συνάρτηση n+1 μεταβλητών ως εξής:

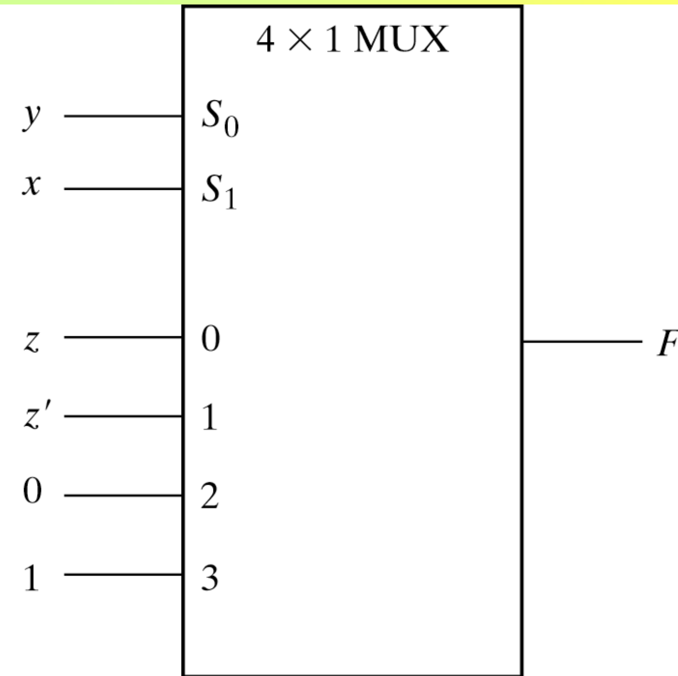
1. Βάζουμε τις n μεταβλητές στις εισόδους επιλογής.
2. Χρησιμοποιούμε την τελευταία μεταβλητή για τις εισόδους.

Παράδειγμα

F(x,y,z) = Σ(1,2,6,7)

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Υλοποίηση Συναρτήσεων Boole

Αλγόριθμος υλοποίησης συνάρτησης n μεταβλητών με χρήση πολυπλέκτη:

1. Χρησιμοποιούμε πολυπλέκτη 2^{n-1}-σε-1 με n-1 γραμμές επιλογής.

1. Δημιουργούμε τον πίνακα αλήθειας της συνάρτησης.

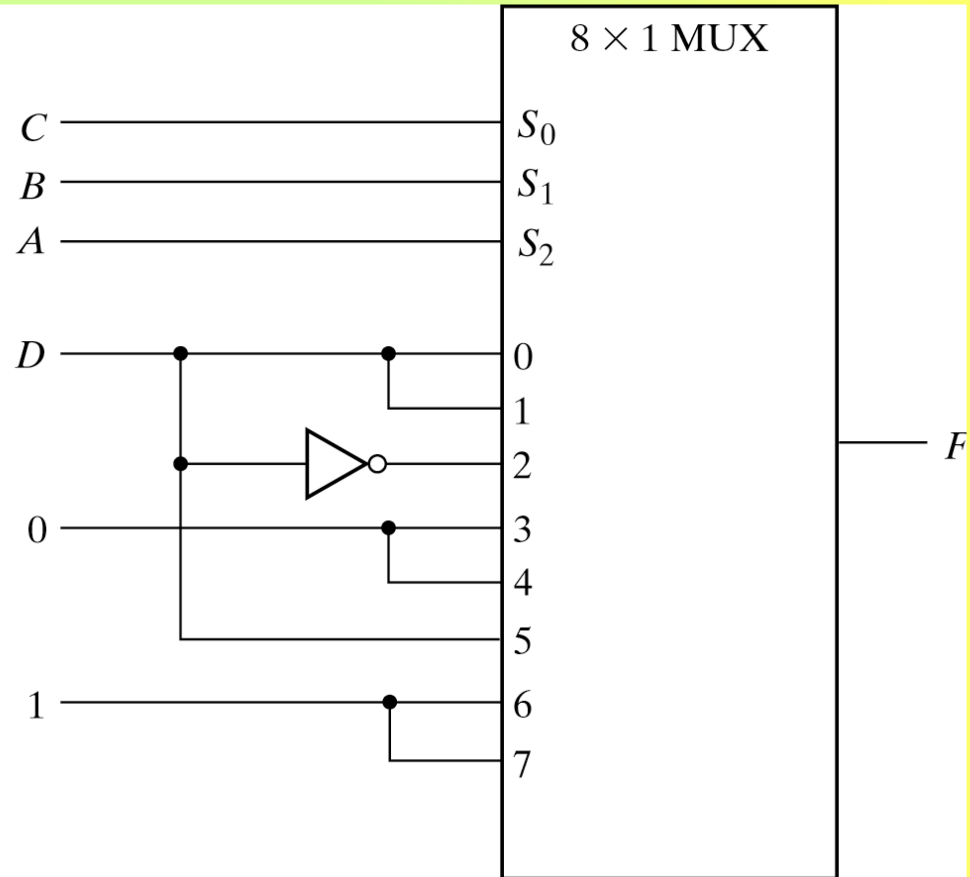
2. Συνδέουμε τις *n* - 1 περισσότερο σημαντικές μεταβλητές στις γραμμές επιλογής και κρατάμε τη δεξιότερη (λιγότερο σημαντική).

3. Για κάθε συνδυασμό των μεταβλητών των γραμμών επιλογής, εκφράζουμε την έξοδο ως συνάρτηση της τελευταίας μεταβλητής.

Παράδειγμα

F(A,B,C,D) = Σ(1,3,4,11,12,13,14,15)

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

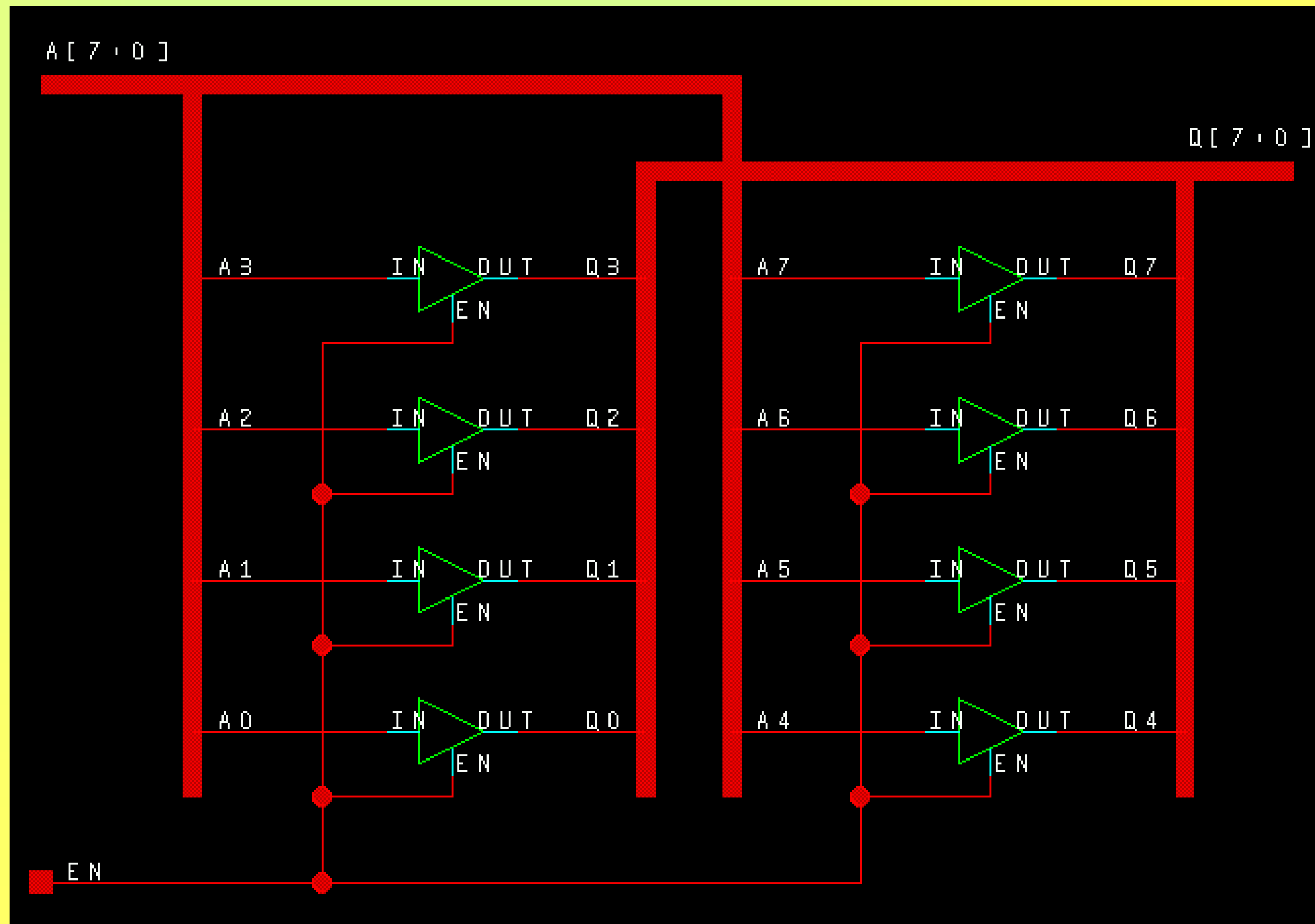


Στοιχεία τριών καταστάσεων

- Ο Α και ο Β θέλουν να διαμοιραστούν την ίδια αρτηρία. Ο Α και ο Β απαγορεύεται να οδηγήσουν μαζί την αρτηρία. Πως μπορώ να αποκλείσω αυτή τη πιθανότητα ?
- Με ένα πολυπλέκτη επιτρέπω είτε στον Α είτε στο Β να οδηγούν την αρτηρία !
- Καλή λύση, αν εκ των προτέρων ξέρω πόσοι και ποιοι θα διαμοιράζονται την αρτηρία !
- Ξέρει ο κατασκευαστής του PC σας πόσους δίσκους θα βάλετε ? Πόσα περιφερειακά θα προσθέσετε στην αρτηρία USB ?
- Τα στοιχεία τριών καταστάσεων μας επιτρέπουν να προσθέτουμε συσκευές σε μια αρτηρία, προσφέροντάς μας τη δυνατότητα να αποκόπτουμε τη συσκευή μας από την αρτηρία στους χρονικούς κύκλους που δε της επιτρέπεται να οδηγήσει την αρτηρία.

Στοιχεία τριών καταστάσεων

In	En	Out
0	0	Z
1	0	Z
0	1	0
1	1	1



ΑΚΟΛΟΥΘΙΑΚΑ ΣΤΟΙΧΕΙΑ

Συνδυαστικά vs Ακολουθιακά

– Συνδυαστικά : οι έξοδοι εξαρτώνται σε κάθε χρονική στιγμή αποκλειστικά και μόνο από τις εισόδους που εφαρμόζονται. Δεν εξαρτώνται ούτε από τη σειρά εφαρμογής, ούτε από τη κατάσταση του κυκλώματος πριν εφαρμοστούν.

– Ακολουθιακά : οι έξοδοι κάθε χρονική στιγμή εξαρτώνται όχι μόνο από τις τιμές των εισόδων εκείνη τη στιγμή αλλά και από τις τιμές των εισόδων όλες τις προηγούμενες χρονικές στιγμές. Οι τελευταίες έχουν επιβάλλει μια *κατάσταση* στο κύκλωμα.

– Η κατάσταση αποθηκεύεται σε *στοιχεία με ικανότητα μνήμης.* Τα στοιχεία αυτά ονομάζονται ακολουθιακά στοιχεία. Ένα κύκλωμα που περιέχει ακολουθιακά στοιχεία είναι ένα ακολουθιακό κύκλωμα.

– Υπάρχουν 2 μεγάλες κατηγορίες ακολουθιακών κυκλωμάτων : *σύγχρονα και ασύγχρονα* κυκλώματα. Στα σύγχρονα οι αλλαγές της κατάστασης γίνονται σε διακριτές στιγμές χρόνου.

Σύγχρονα vs Ασύγχρονα

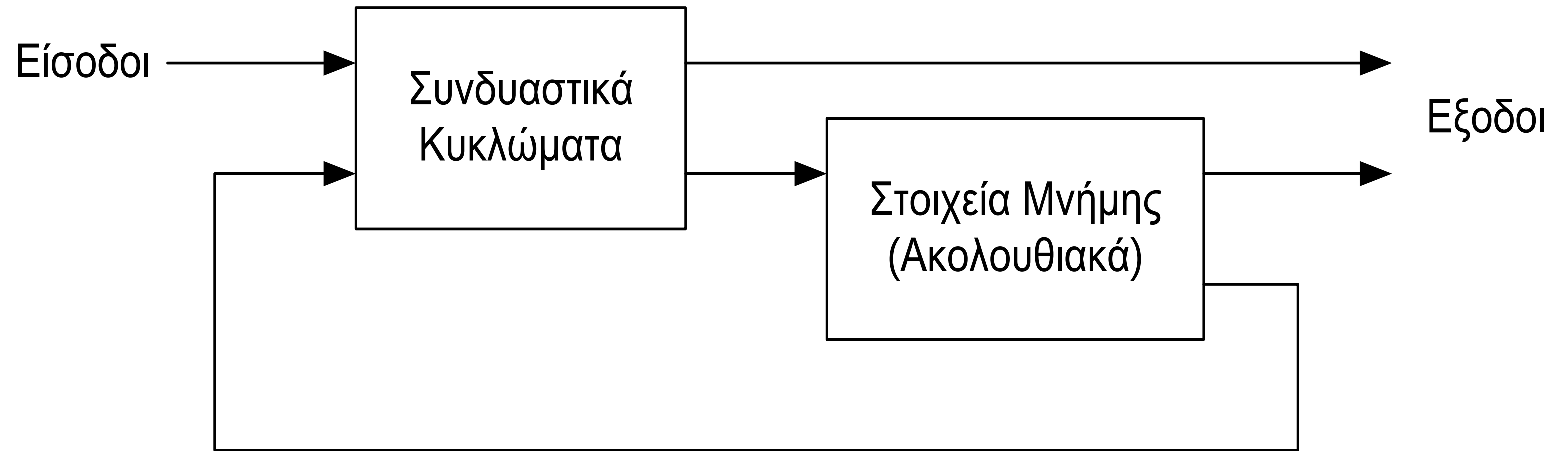
Σύγχρονα

- Οι τιμές σε συγκεκριμένες (διακριτές) τιμές χρόνου το περιγράφουν πλήρως
- Συγχρονισμός επιτυγχάνεται μέσω ενός σήματος χρονισμού, γνωστό ως ρολόι που έχει μια περιοδική παλμοσειρά.
- Δε παρουσιάζουν αστάθεια.

Ασύγχρονα

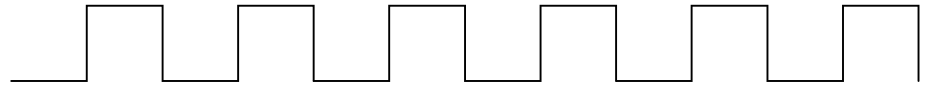
- Οι τιμές εξαρτώνται από τη σειρά εναλλαγής των σημάτων εισόδου. Νέα τιμή μπορεί να προκύψει ανά πάσα χρονική στιγμή.
- Ένα συνδυαστικό κύκλωμα με ανατροφοδότηση (feedback) είναι ένα ασύγχρονο ακολουθιακό κύκλωμα.
- Αστάθεια.

Γενικό μοντέλο ενός πλήρους κυκλώματος

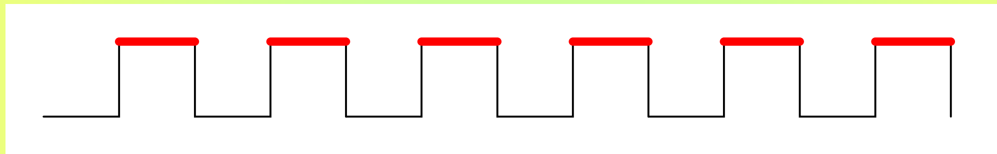


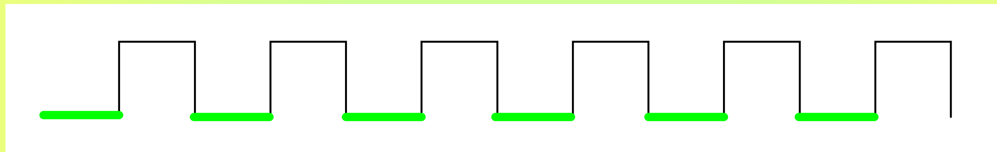
Ακολουθιακά στοιχεία / στοιχεία μνήμης - γενικά

– Εστω το σήμα χρονισμού :



– Υπάρχουν δύο (2) κατηγορίες :
  - Αυτά που είναι ενεργά βάσει μιας στάθμης δυναμικού :





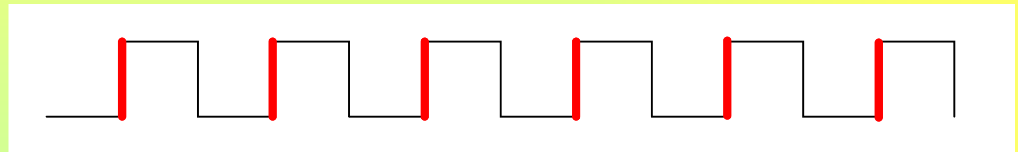
  - Το στοιχείο αυτό μπορεί να αλλάζει κατάσταση διαρκώς όσο το σήμα χρονισμού βρίσκεται στην ενεργή στάθμη.

  - Τα στοιχεία αυτής της κατηγορίας ονομάζονται μανταλωτές ***(latches)***

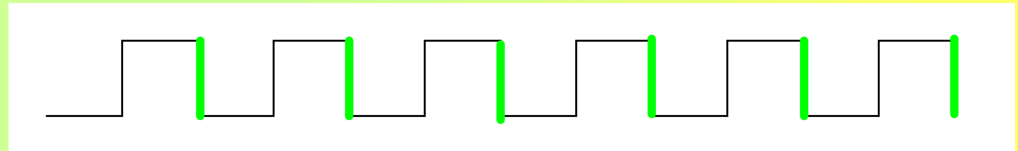
Ακολουθιακά στοιχεία / στοιχεία μνήμης – γενικά - 2

- Αυτά που είναι ενεργά μόνο κατά την μία ακμή αλλαγής δυναμικού

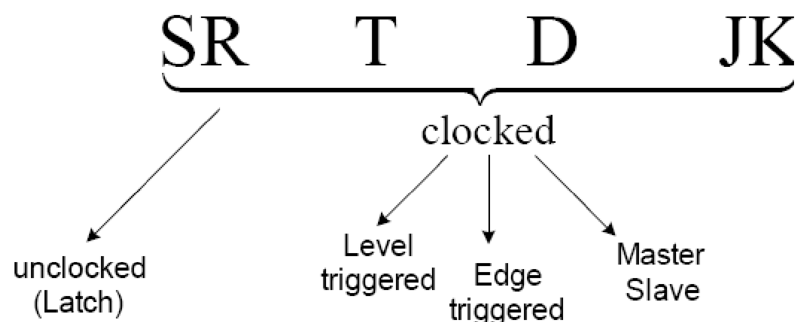
Θετική ακμοπυροδότηση



Αρνητική ακμοπυροδότηση

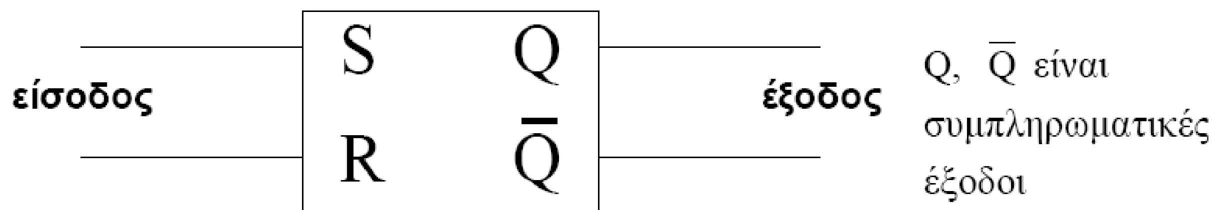


- Τα στοιχεία αυτής της κατηγορίας ονομάζονται ***flip-flops***



Τα flip-flops αποτελούν τα βασικά δομικά στοιχεία για το σχεδιασμό των ακολουθιακών κυκλωμάτων.

Το SR (Set-Reset) flip-flop (ff)



Πίνακας αληθείας του SR-ff

S	R	Q_{n+1}	
			← επόμενη κατάσταση
0	0	Q_n	← παρούσα κατάσταση
0	1	0	
1	0	1	
1	1	-	ακαθόριστη έξοδος (μη επιτρεπτή περίπτωση)

Για την υλοποίηση του SR-ff δημιουργούνται ο εκτεταμένος πίνακας αληθείας και οι πίνακες Karnaugh, όπου το Qn (παρούσα κατάσταση εξόδου) χρησιμοποιείται ως μεταβλητή εισόδου:

R	S	Q_n	Q_{n+1}	
0	0	0	0	Μνήμη
0	0	1	1	Μνήμη
0	1	0	1	Set
0	1	1	1	Set
1	0	0	0	Reset
1	0	1	0	Reset
1	1	0	x	
1	1	1	x	

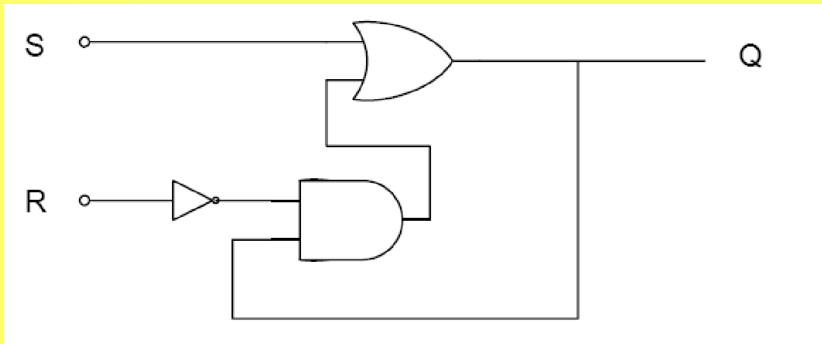
Πίνακας-Κ

Q_{n+1} R \ SQ_n	00	01	11	10
0	0	1	1	1
1	0	0	X	X

$$Q_{n+1} = S + \overline{R}Q_n$$

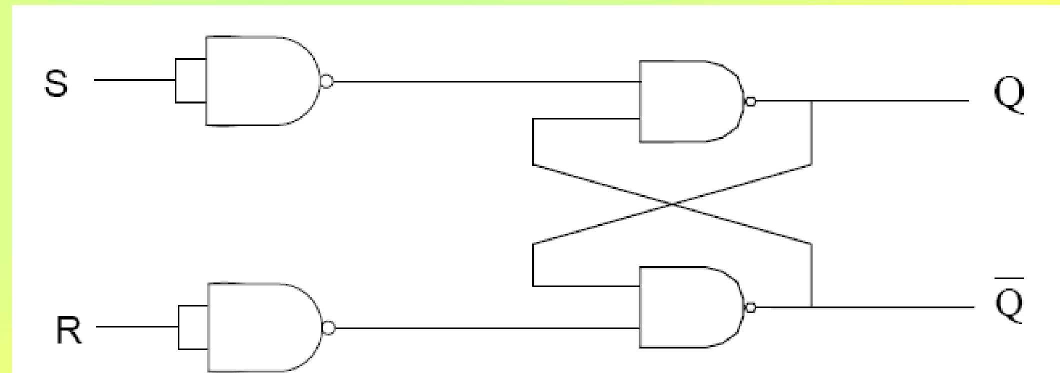
Χαρακτηριστική εξίσωση

Το κύκλωμα που υλοποιεί την παραπάνω σχέση είναι:



Χρησιμοποιώντας το θεώρημα De Morgan, η σχέση για σχεδιασμό με πύλες NAND έχει ως εξής:

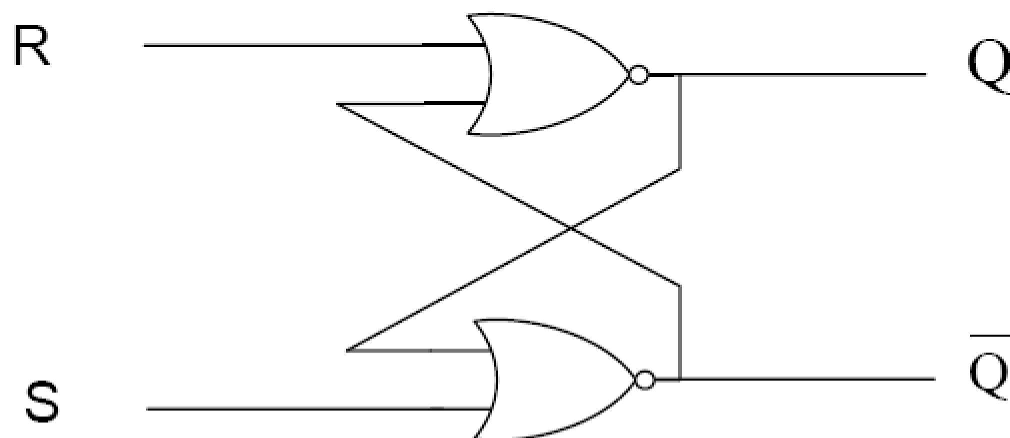
$$Q_{n+1} = \overline{\overline{S} \cdot \overline{\overline{R} Q_n}}$$



Χρησιμοποιώντας έναν εκτεταμένο πίνακα αληθείας και τους πίνακες-Κ που συσχετίζουν τα $\overline{Q}_{n+1}$, $\overline{Q}_n$, R και S προκύπτει μια άλλη χαρακτηριστική εξίσωση η οποία δίνεται από τη σχέση:

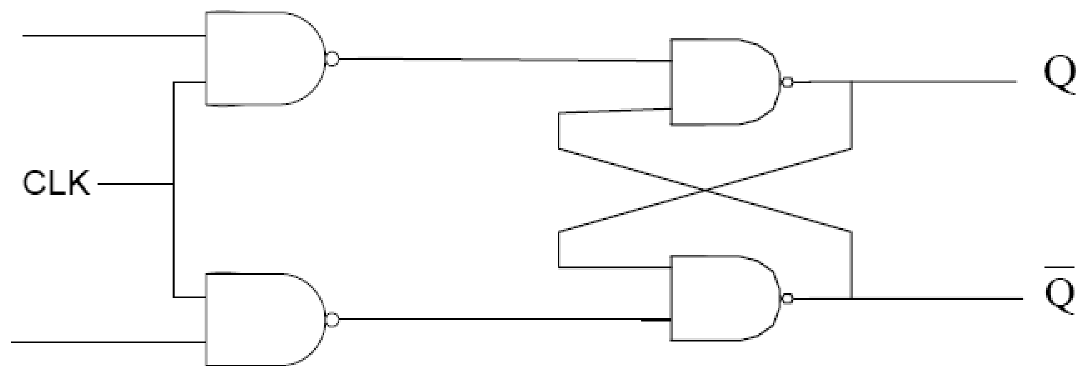
$$\overline{Q}_{n+1} = R + \overline{S}\,\overline{Q}_n$$

από την οποία εξάγεται η σχέση για την υλοποίηση του ff με πύλες NOR



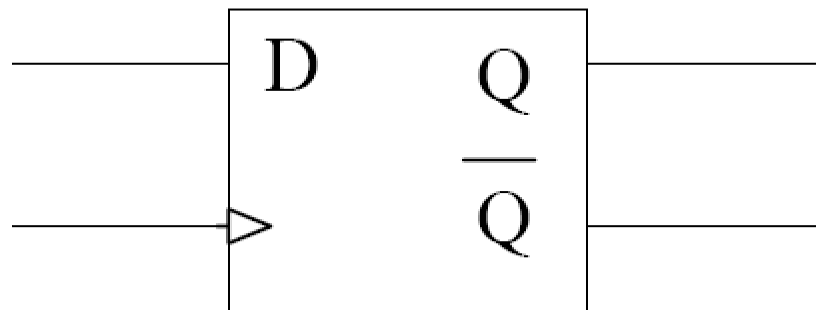
$$Q_{n+1} = \overline{R + \overline{S + Q_n}}$$

To Clocked SR - ff



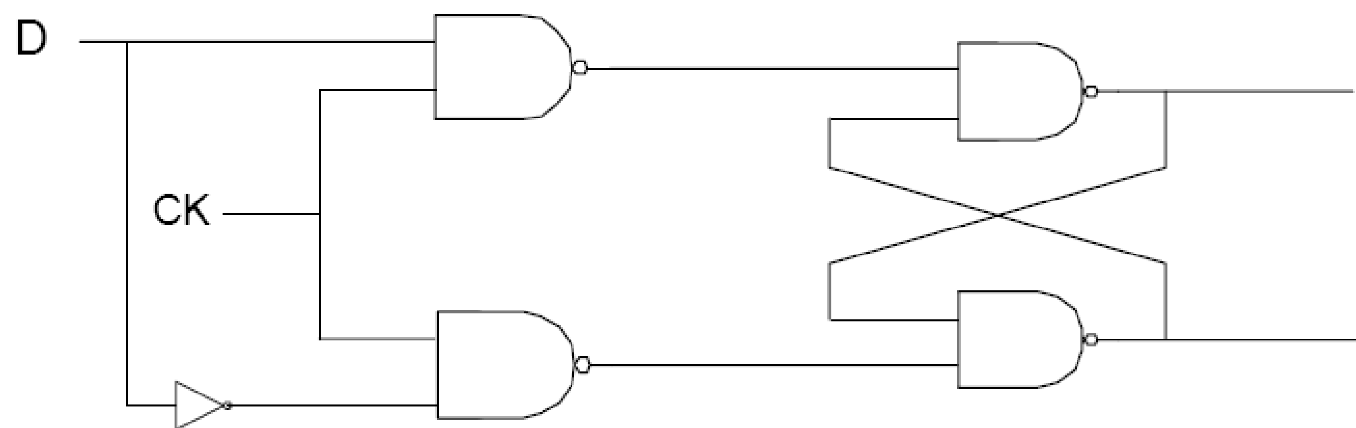
Το σήμα του ρολογιού δρα ως ένα σήμα που ενεργοποιεί το SR-ff και επομένως οι έξοδοί του μπορεί να αλλάξουν μόνο όταν ο παλμός του ρολογιού είναι 1.

Flip – Flop τύπου D

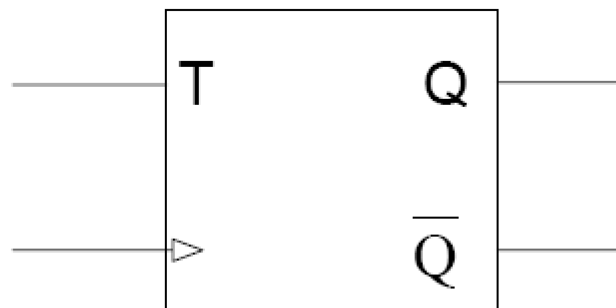


Ck	D	Q_{n+1}
0	0	Q_n
0	1	Q_n
1	0	0
1	1	1

Υλοποίηση:

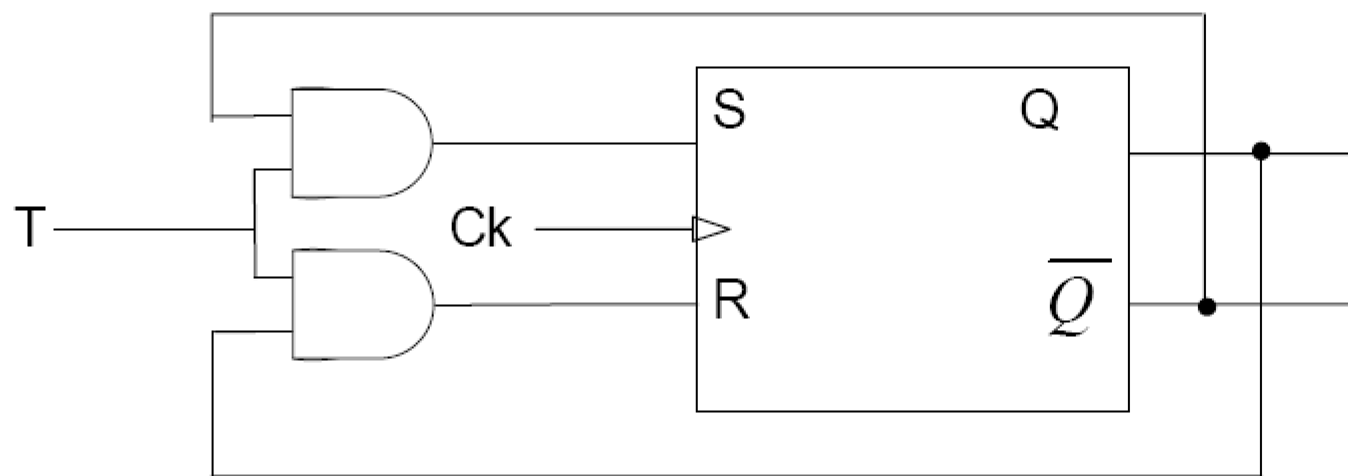


Flip – Flop τύπου T (Toggle ff)



Ck	T	Q_{n+1}
0	0	Q_n
0	1	Q_n
1	0	Q_n
1	1	$\overline{Q_n}$

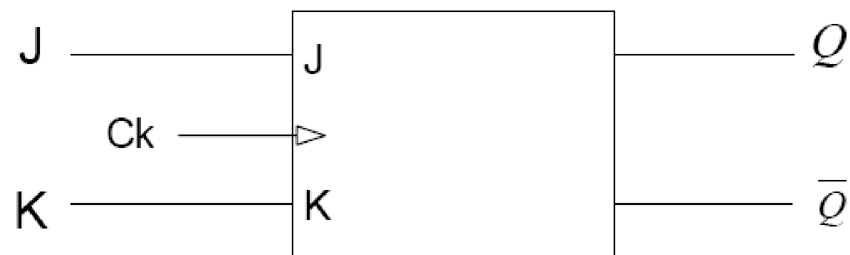
Υλοποίηση:



$$S = T\overline{Q}$$

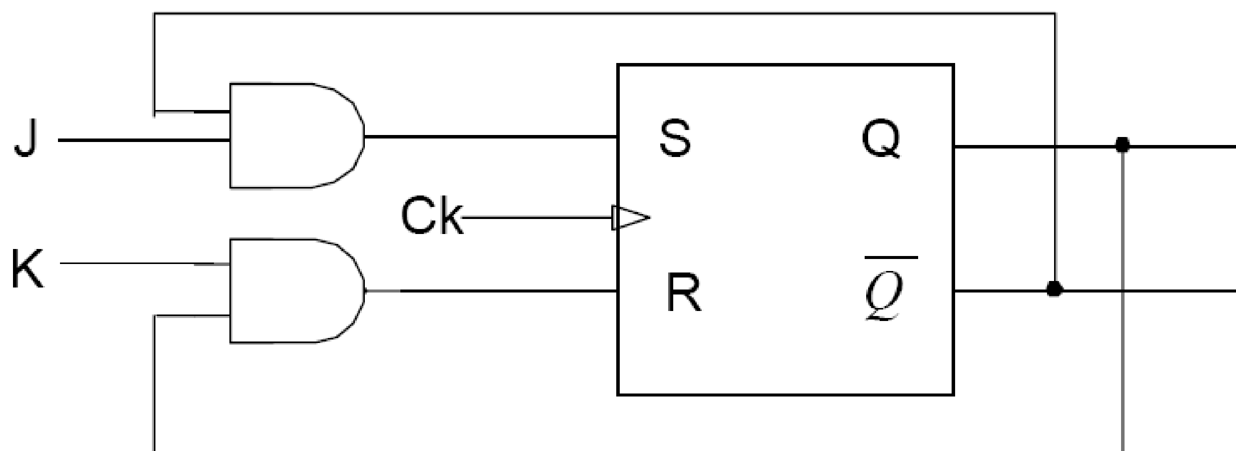
$$R = TQ$$

Flip – Flop τύπου JK



Ck	J	k	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q}_n$

Υλοποίηση:



$$S = J\overline{Q}$$

$$R = KQ$$

Αλλα είδη flip - flop

- Το D flip flop είναι το πιο οικονομικό από άποψης απαιτούμενων πυλών.
- Ωστόσο δεν είναι το μόνο που μπορούμε να φτιάξουμε. Τα πιο διαδεδομένα άλλα είδη flip flop είναι :
  - Το Master / Slave S-R Flip – Flop
  - Το Master / Slave J-K Flip – Flop
  - Το ακμοπυροδότητο J-K Flip – Flop
  - Το T Flip – Flop
  - Το Scan Flip – Flop
- Σε καθένα από αυτά, μπορούμε να προσθέτουμε άμεσες εισόδους είτε σύγχρονες είτε ασύγχρονες.
- Η συλλογή όλων αυτών των ακολουθιακών στοιχείων αποτελεί μια βιβλιοθήκη στοιχείων με την οποία φτιάχνουμε τα ακολουθιακά κυκλώματά μας.

Scan Flip Flop (σάρωσης)

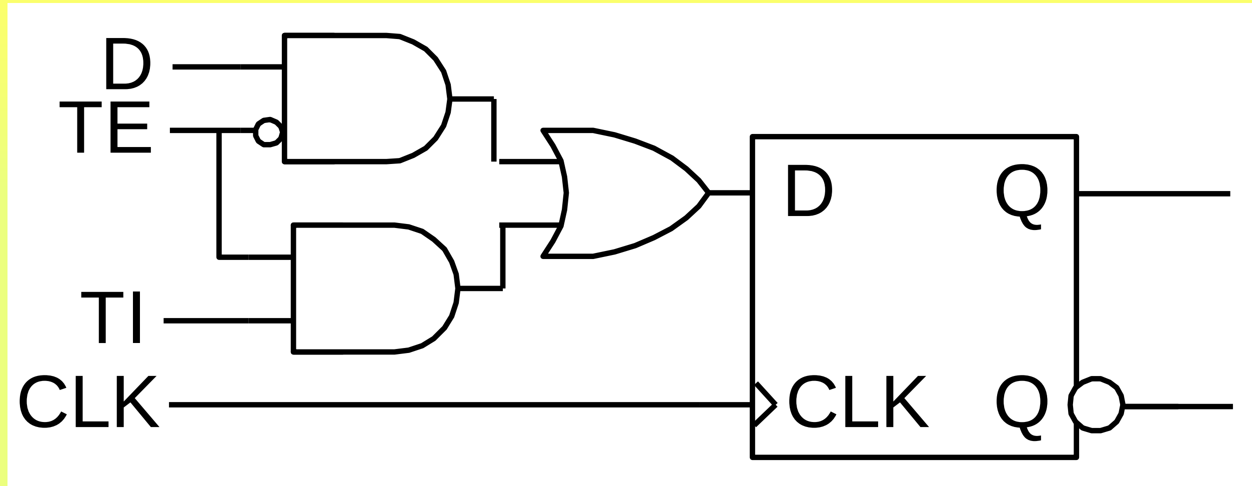
– Στα σημερινά ολοκληρωμένα κυκλώματα υπάρχουν χιλιάδες έως εκατοντάδες χιλιάδες FF.

– Όταν θέλουμε να δοκιμάσουμε αν ένα τέτοιο κύκλωμα δουλεύει σωστά θα πρέπει να μπορούμε να βάζουμε τα FFς του σε συγκεκριμένες καταστάσεις αλλά και να διαβάζουμε τις εξόδους τους.

– Αυτό δε μπορεί να γίνει (ή είναι εξαιρετικά χρονοβόρο) χρησιμοποιώντας τις εισόδους / εξόδους λειτουργίας του FF.

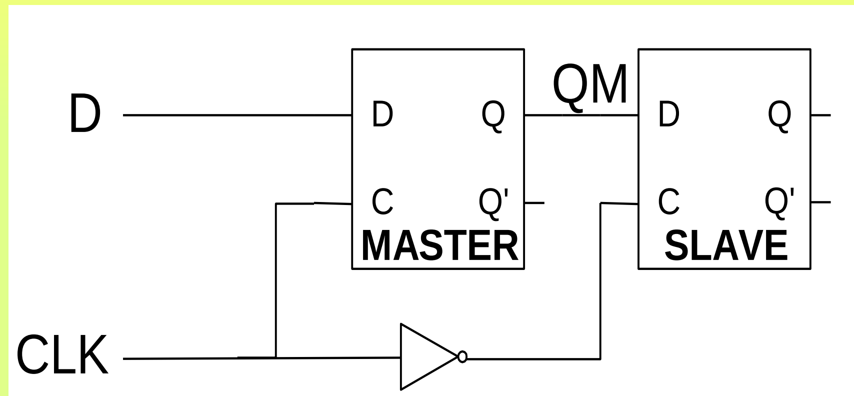
– Σκοπός του scan FF είναι να παρέχει εναλλακτική είσοδο / έξοδο δεδομένων που τη χρησιμοποιούμε κατά τη διάρκεια των δοκιμών μόνο.

Scan D FF



- Ισοδύναμο με ένα DFF και ένα πολυπλέκτη στην είσοδο με σήμα επιλογής το ~TE (Test Enable).
- Για κανονική λειτουργία TE = 0 και παίρνουμε ένα D FF.
- Όταν TE = 1, η είσοδος D, οδηγείται από το TI (Test Input).
- Οι επιπλέον είσοδοι χρησιμοποιούνται για τη σύνδεση όλων των FF του κυκλώματος σε μια αλυσίδα σάρωσης (scan chain).

Αρνητικά ακμοπυροδότητο D flip-flop

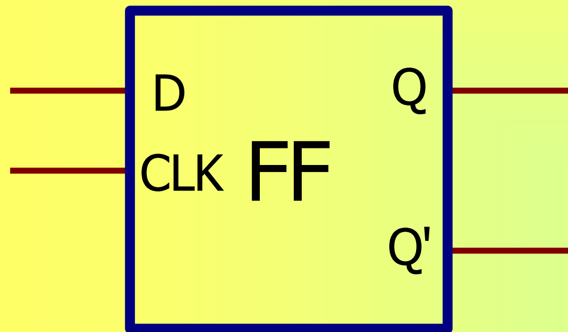


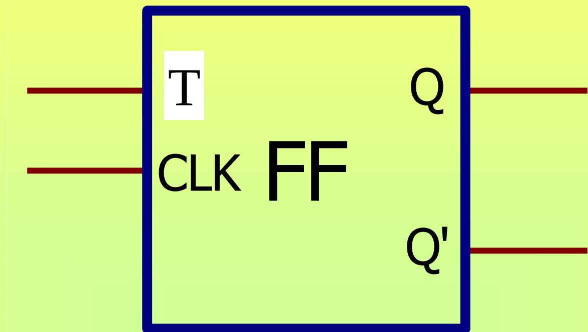
D	CLK	Q	Q'
0	↓	0	1
1	↓	1	0
X	0	Q(t-1)	Q'(t-1)
X	1	Q(t-1)	Q'(t-1)

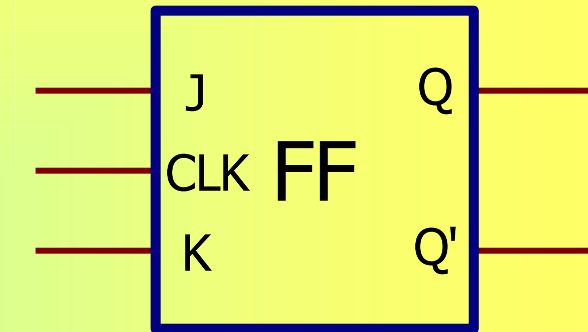
Τι θα πρέπει να θυμάστε

- Ακμοπυροδότητα και πυροδότητα με τιμή (level vs edge triggered) στοιχεία.
- Άμεσες είσοδοι για Reset και Preset.
- Το D FF είναι το πιο οικονομικό και γι’ αυτό χρησιμοποιείται κατά κόρο.
- Άλλα χρησιμοποιούμενα FFs είναι τα J-K και T.
- Κάθε ff έχει μια χαρακτηριστική εξίσωση που προκύπτει από απλοποίηση με πίνακες –K.

Πίνακες διεγέρσεως FF







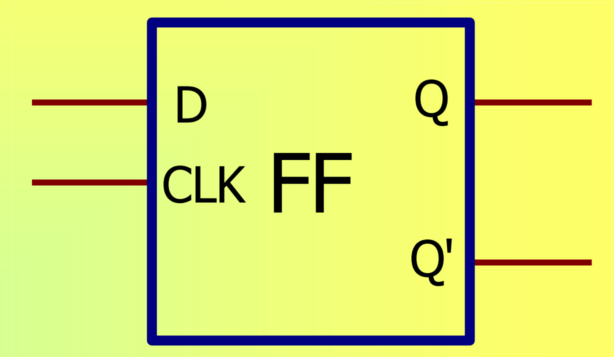
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

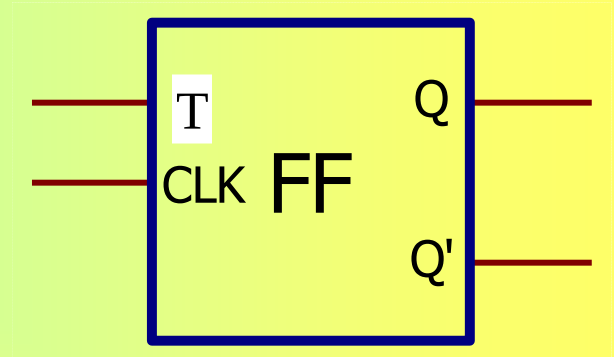
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Χαρακτηριστικοί πίνακες FF

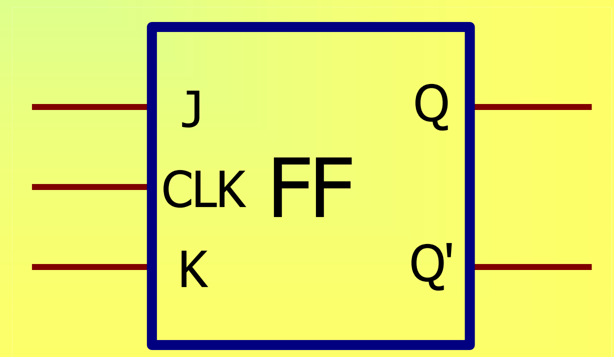
CLK	D	Q_{n+1}	
↑	1	1	Load "1" (Set)
↑	0	0	Load "0" (Reset)



CLK	T	Q_{n+1}	
↑	0	Q_n	Μνήμη/Παραμονή
↑	1	Q'_n	Εναλλαγή (Toggle)



CLK	J	K	Q_{n+1}	
↑	0	0	Q_n	ηρεμία *(μνήμη)*
↑	1	0	1	Load "1" (Set)
↑	0	1	0	Load "0" (Reset)
↑	1	1	Q'_n	Toggle



Σχεδιασμός D_Flip_Flop σε VHDL.

Xilinx - ISE - C:\Desktop\D_Flip_Flop\D_Flip_Flop.ise - [D_Flip_Flop.vhd]

File Edit View Project Source Process Window Help

Sources

Sources for: Synthesis/Impleme Number of: LUTs

Hierarchy Resources Preserve

- D_Flip_Flop
- xc2s50-5t...
  - D...

Sources Snapshots Libraries

Processes

Processes:

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
  - Create Timing Constraints
  - Assign Package Pins
  - Create Area Constraints
  - Edit Constraints (Text)
- Synthesize - XST
- Implement Design
- Generate Programming File

Processes

```
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity D_Flip_Flop is
    Port ( rst : in  STD_LOGIC;
           D : in  STD_LOGIC;
           clk  : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end D_Flip_Flop;

architecture Behavioral of D_Flip_Flop is

begin
process(rst,clk)
begin
if (rst='1') then Y<='1';
else if clk='1' then Y<=D;
end if;
end if;
end process;
```

D_Flip_Flop.vhd Design Summary

```
Started : "Generate Programming File".

Process "Generate Programming File" completed successfully
```

Transcript

Console Errors Warnings Tcl Console Find in Files

Ln 46 Col 13 CAPS NUM SCRL VHDL

start Ergastiriaki askisi 7 - ... ergasthriaki askisi 6 a... Xilinx - ISE - C:\Deskt... axitektoniki EN 9:37 μμ

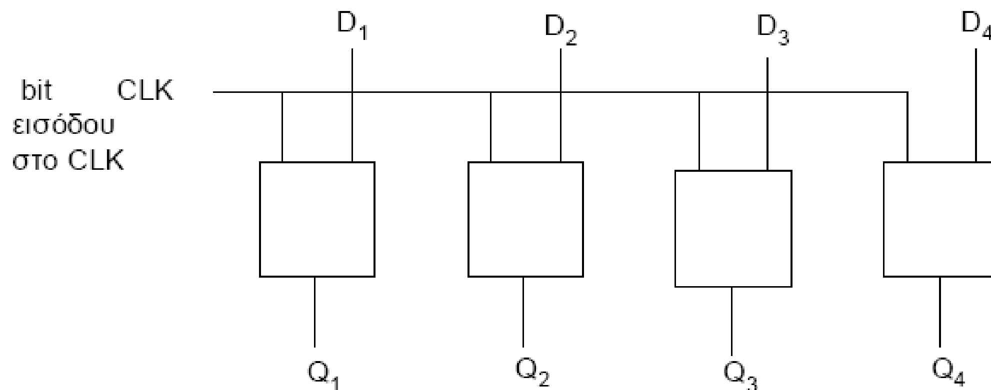
Εφαρμογές ff

- Τυπικές εφαρμογές των ff είναι:

Απλοί καταχωρητές

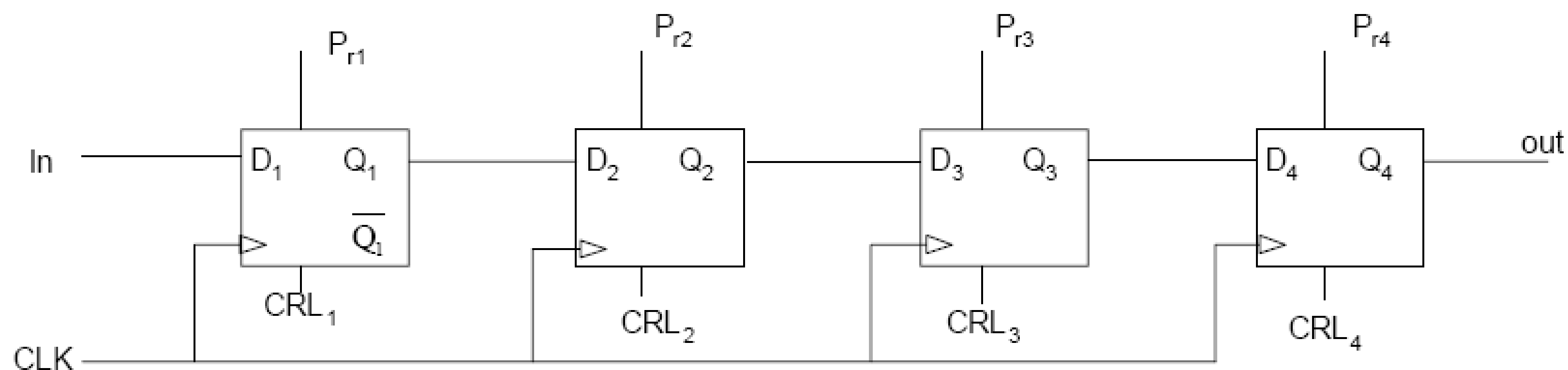
Κυκλώματα καταχωρητών ολίσθησης

Μετρητές



Καταχωρητές ολίσθησης

Για παράδειγμα καταχωρητής ολίσθησης 4 bit χρησιμοποιώντας D-ff



DATA IN	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$	$\mathbf{Q_4}$
1	0	0	0	0
1	1	0	0	0
0	1	1	0	0
0	0	1	1	0
1	0	0	1	1
1	1	0	0	1

Ο παραπάνω καταχωρητής είναι γνωστός και ως καταχωρητής SISO (Serial In Serial Out). Εάν σ΄ ένα SISO καταχωρητή το Q1 είναι το πιο σημαντικό ψηφίο και το Q4 το πιο χαμηλής σημαντικότητας ψηφίο (MSB και LSB αντίστοιχα), τότε η μετατόπιση γίνεται προς τα δεξιά.

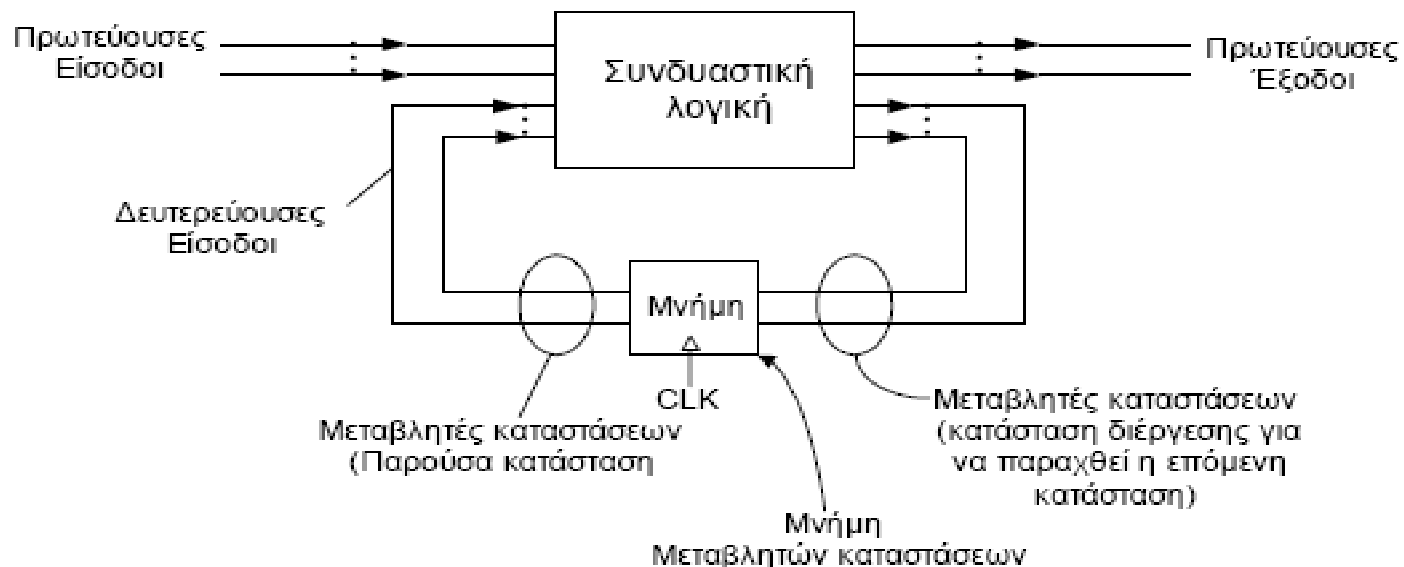
Στην αντίθετη περίπτωση, δηλαδή το Q4 → MSB και το Q1 → LSB, τότε η μετατόπιση γίνεται προς τα αριστερά.

Σημείωση: Κάθε είσοδος 0 στον καταχωρητή μετατόπισης έχει ως αποτέλεσμα

- Τη διαίρεση με το 2 εάν είναι ο καταχωρητής μετατόπισης προς τα δεξιά και
- τον πολλαπλασιασμό με το 2 εάν είναι καταχωρητής μετατόπισης προς τα αριστερά

Εισαγωγή στο σχεδιασμό ψηφιακών κυκλωμάτων με διαγράμματα καταστάσεων

- Μπορούν να χρησιμοποιηθούν για το σχεδιασμό σύγχρονων ακολουθιακών κυκλωμάτων
- Γενική μορφή σύγχρονου ακολουθιακού κυκλώματος που χρησιμοποιείται σε διαγράμματα καταστάσεων (Δ.Κ.)

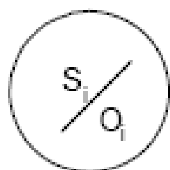


Ένα σύστημα με ‘N’ μεταβλητές καταστάσεων χρειάζεται N flip-flops για μνήμη.

Διαγράμματα καταστάσεων

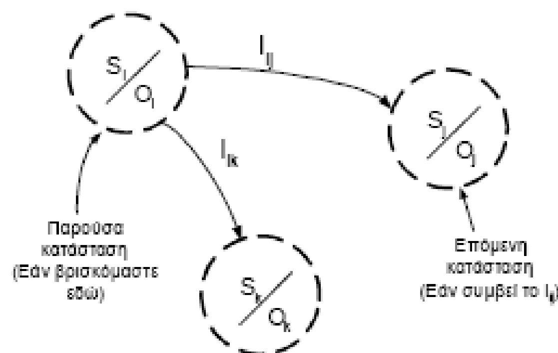
- Είναι η γραφική παράσταση της λειτουργίας ενός ακολουθιακού κυκλώματος

Στοιχείο κατάστασης:



Όπου S_i είναι η περιγραφή της κατάστασης i και O_i είναι οι έξοδοι που σχετίζονται με την κατάσταση S_i

Στοιχείο μετάβασης καταστάσεων:



I_{ij} : είναι συνθήκες που προκαλούν τη μετάβαση από την κατάσταση Si → S_j

Η παραπάνω γραφική αναπαράσταση των διαγραμμάτων κατάστασης είναι γνωστή και ως παράσταση Moore.

Πίνακες Καταστάσεων

Είναι η παράσταση του διαγράμματος καταστάσεων υπό μορφή πίνακα.

Σ' αυτόν περιλαμβάνονται η παρούσα κατάσταση (PS) η επόμενη κατάσταση (NS), οι συνθήκες εισόδου που προκαλούν τη μετάβαση και οι αντίστοιχες τιμές εξόδου.

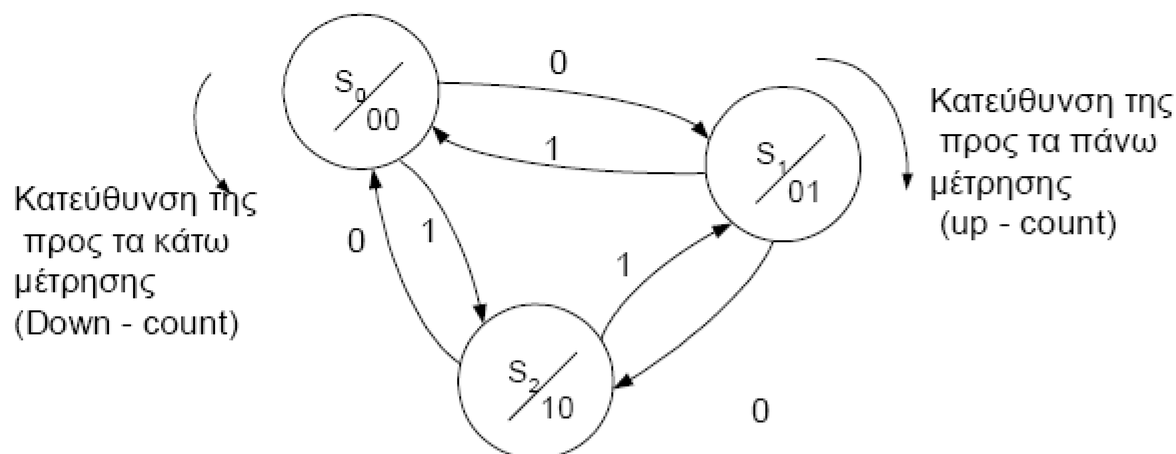
Παράδειγμα 2.3

Μετρητής MOD-3 (UP/DOWN)

είσοδοι : DIR = 1 : Μετράει μπρος τα κάτω (Down)

DIR = 0 : Μετράει μπρος τα επάνω (Up)

έξοδοι : A,B



Είσοδοι (DIR)	PS	MS	Έξοδοι A	B
0	S_0	S_1	0	0
1	S_0	S_2	0	0
0	S_1	S_2	0	1
1	S_1	S_0	0	1
0	S_2	S_0	1	0
1	S_2	S_1	1	0

1η Μορφή

PS	NS DIR=0	NS DIR=1	Έξοδοι A	B
S_0	S_1	S_2	0	0
S_1	S_2	S_0	0	1
S_2	S_0	S_1	1	0

2η Μορφή

Πίνακας διέγερσης και εξισώσεις διέγερσης

Ο πίνακας διέγερσης περιλαμβάνει τις τιμές της εισόδου που απαιτούνται για κάθε μετάβαση μεταξύ των καταστάσεων, για επιλεγμένο τύπο flip-flop. Ο προηγούμενος πίνακας καταστάσεων δημιουργήθηκε ανεξάρτητα από τον τύπο του flip-flop, ή τον τύπο της συνδυαστικής λογικής που επρόκειτο να χρησιμοποιηθεί. Άπαξ και αποφασισθεί ο τύπος του flip-flop που πρόκειται να χρησιμοποιηθεί, προσδιορίζονται οι μεταβλητές καταστάσεων και κατασκευάζεται ο πίνακας διέγερσης.

Σ' ένα τυπικό ακολουθιακό κύκλωμα θα μπορούσαν να χρησιμοποιηθούν είτε D-ffs είτε JK-ffs.

D-ffs	**JK-ffs**
Απλούστερη επεξεργασία Κατά το σχεδιασμό	Λιγότερο σύνθετο τελικό κύκλωμα

Πρόβλημα

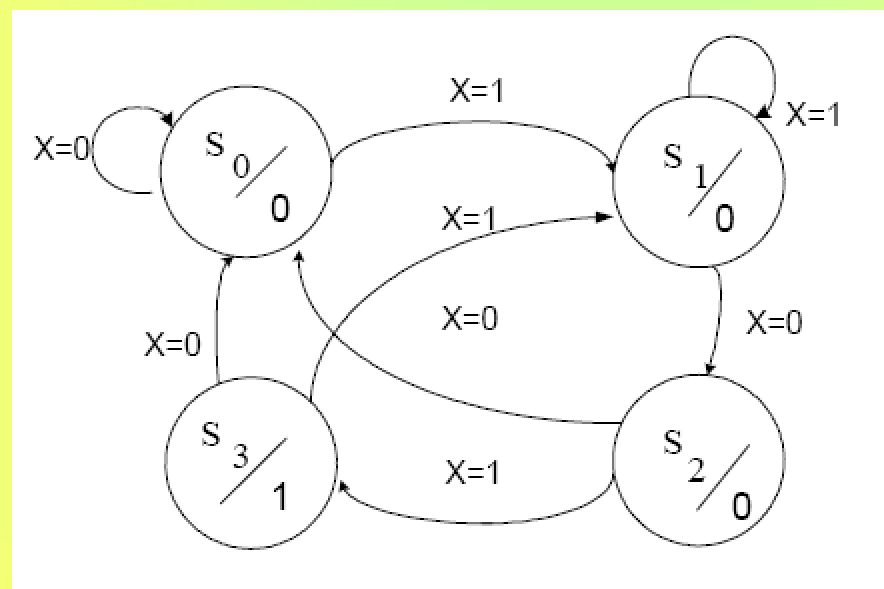
Να σχεδιαστεί ένα ακολουθιακό κύκλωμα που ανιχνεύει την ακολουθία 1 0 1 από ένα σύνολο δυαδικών στοιχείων που εισάγονται σειριακά με ρυθμό 1 bit ανά παλμό ρολογιού (Να χρησιμοποιηθούν JK-ffs)

Λύση:

Το κύκλωμα έχει μια μεταβλητή εισόδου X που παίρνει τις τιμές 0 και 1 και μια μεταβλητή εξόδου Z η οποία γίνεται 1 όταν ληφθεί μια ακολουθία 1 0 1. Κατ' αρχήν δεν είναι φανερό πόσες είναι οι δυνατές καταστάσεις, οι οποίες όμως μπορούν να προσδιοριστούν κάνοντας το διάγραμμα καταστάσεων.

Έστω S_0 η αρχική κατάσταση κατά την οποία κανένα από τα bit εισόδου δεν έχουν φτάσει στη σωστή ακολουθία. Στην κατάσταση S_0, εάν X = 0 παραμένουμε στην κατάσταση S_0, ενώ εάν X = 1 τότε έχουμε το πρώτο bit όπως ακριβώς το αναμένουμε στην επιθυμητή ακολουθία, συνεπώς γίνεται μετάβαση στην κατάσταση S_1. Η έξοδος και στις δυο περιπτώσεις είναι μηδέν αφού δεν έχει ανιχνευθεί ακόμη η πλήρης ακολουθία.

Σκεπτόμενοι κατά τον ίδιο τρόπο εάν είμαστε στη κατάσταση S_1 και X = 0 μεταβαίνουμε στην κατάσταση S_2 , ενώ εάν X = 1 παραμένουμε στην S_1 θεωρώντας ότι το 1 είναι το πρώτο bit της αναμενόμενης ακολουθίας. Εάν είμαστε στην κατάσταση S_2 και X = 1 τότε μεταβαίνουμε στην κατάσταση S_3 όπου η έξοδος Z = 1, ενώ εάν X = 0 επιστρέφουμε στην κατάσταση S_0 . Τέλος από το S_3 για X = 0 επιστρέφουμε στην κατάσταση S_0 , ενώ για X = 1 στην S_1. Σύμφωνα με τα παραπάνω σχεδιάζεται το παρακάτω διάγραμμα καταστάσεων.



Από το διάγραμμα καταστάσεων παρατηρούμε ότι έχουμε τέσσερις καταστάσεις, άρα απαιτούνται 2 μεταβλητές καταστάσεων, η δυο ffs.

Για τα σχεδιασμό του κυκλώματος γίνονται τα παρακάτω βήματα:

1. Πίνακας καταστάσεων

PS	NS x = 0	NS x = 0	Έξοδος Z
S_0	S_0	S_1	0
S_1	S_2	S_1	0
S_2	S_0	S_3	0
S_3	S_0	S_1	1

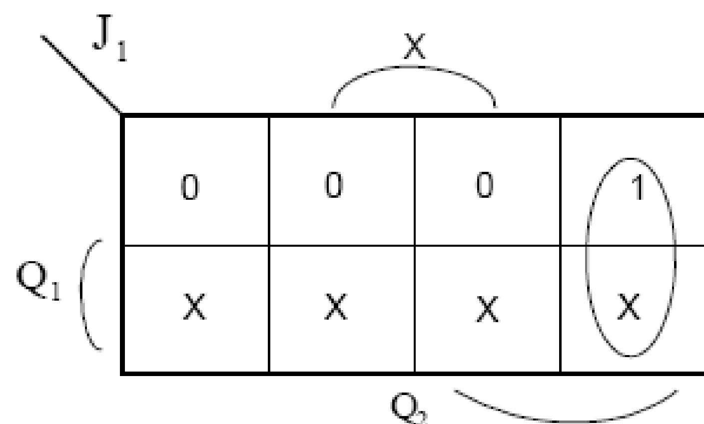
Ορισμός μεταβλητών καταστάσεων

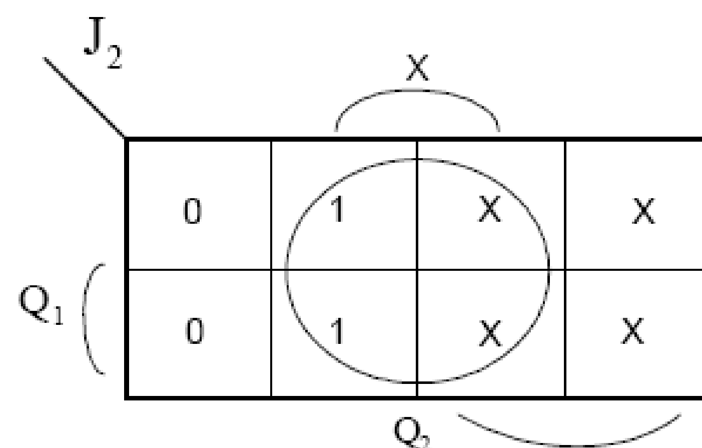
	Q_1	Q_2
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

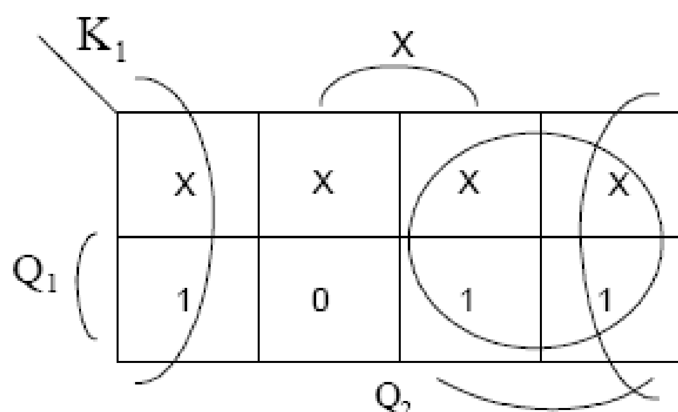
1. Πίνακας διέγερσης

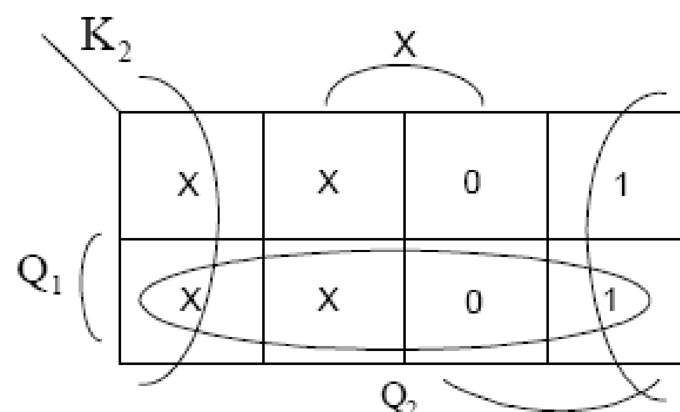
	PS	X=0 NS	EX		X=0 NS	EX		
	Q_1 Q_2	Q_1 Q_2	J_1 K_1	J_2 K_2	Q_1 Q_2	J_1 K_1	J_2 K_2	Z
S_0	0 0	S_0 :0 0	0 X	0 X	S_1: 0 1	0 X	1 X	0
S_1	0 1	S_2 :1 0	X 1	X 1	S_1: 0 1	0 X	1 0	0
S_2	1 0	S_0 :0 0	X 1	0 X	S_3: 1 1	X 0	1 X	0
S_3	1 1	S_0 :0 0	X 1	X 1	S_4: 0 1	X 1	X 0	1

4. Εξισώσεις διέγερσης



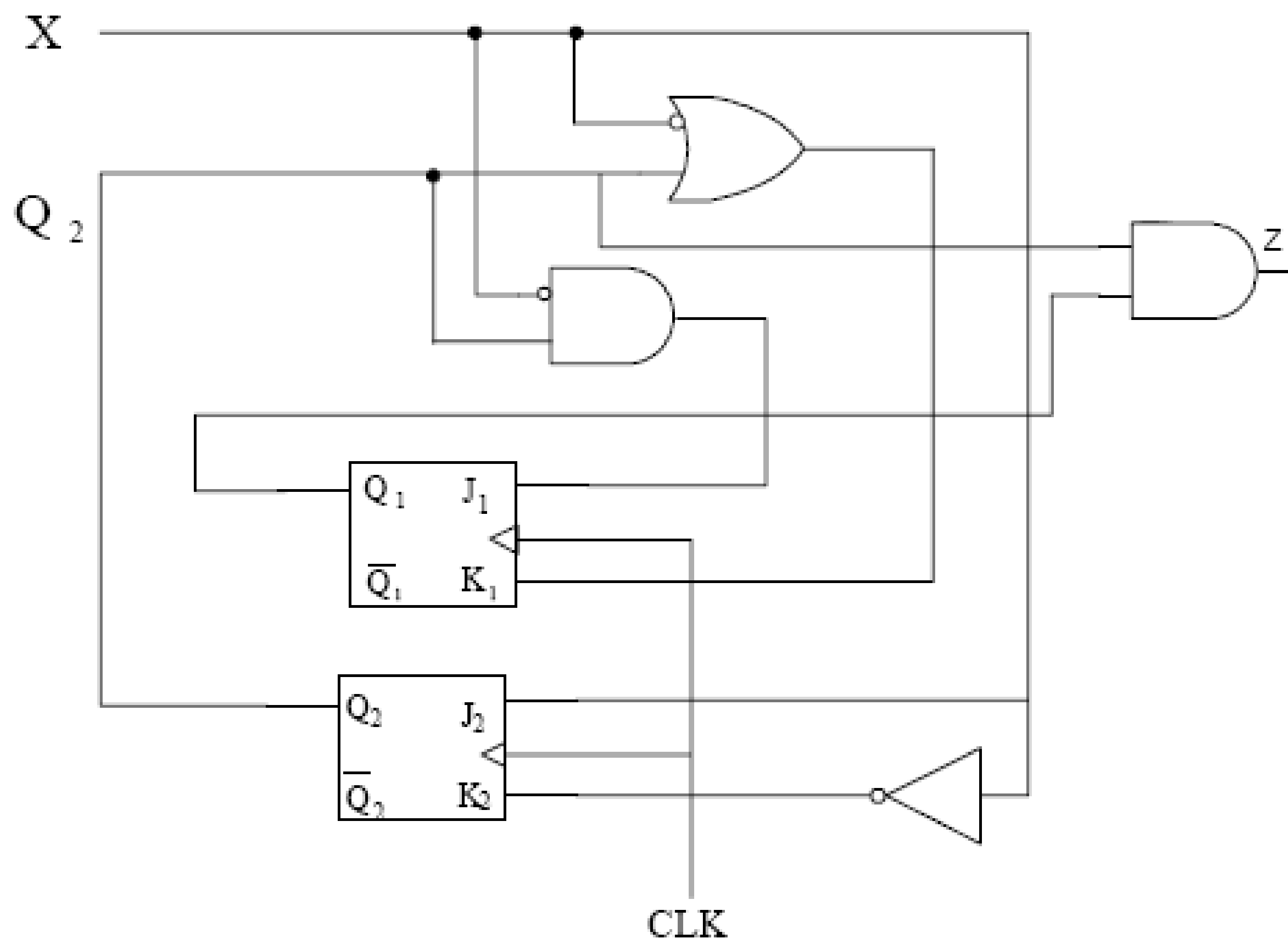






$J_1 = \overline{X}\,Q_2$, $\quad K_1 = \overline{X} + Q_2$, $\quad J2 = X$, $\quad K_2 = QX$

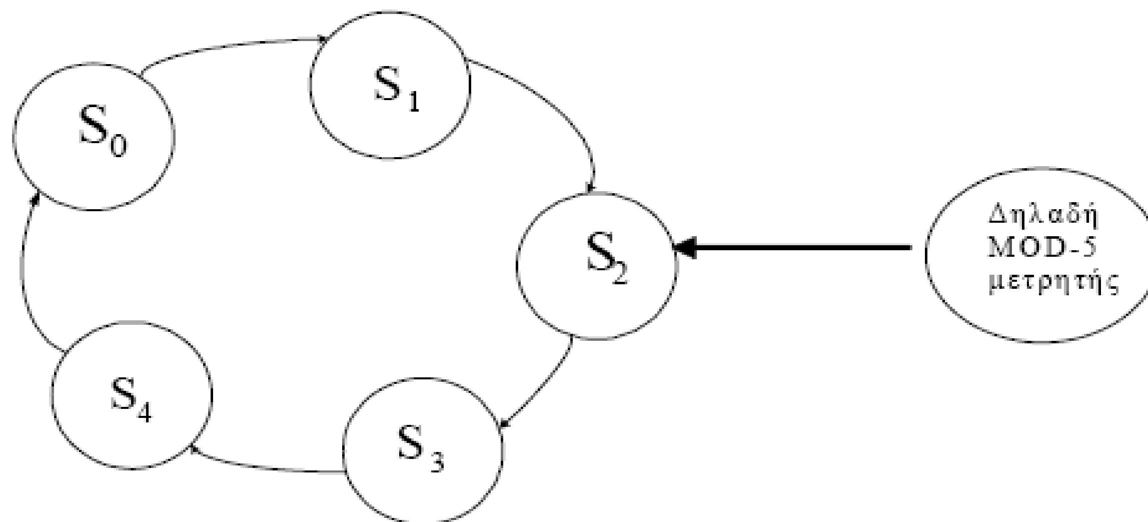
και $\quad Z = Q_1 Q_2$


X
Q 2
Z
Q 1
J1
Q̄1
K1
Q2
J2
Q̄2
K2
CLK

Προβλήματα από καταστάσεις που δεν χρησιμοποιούνται

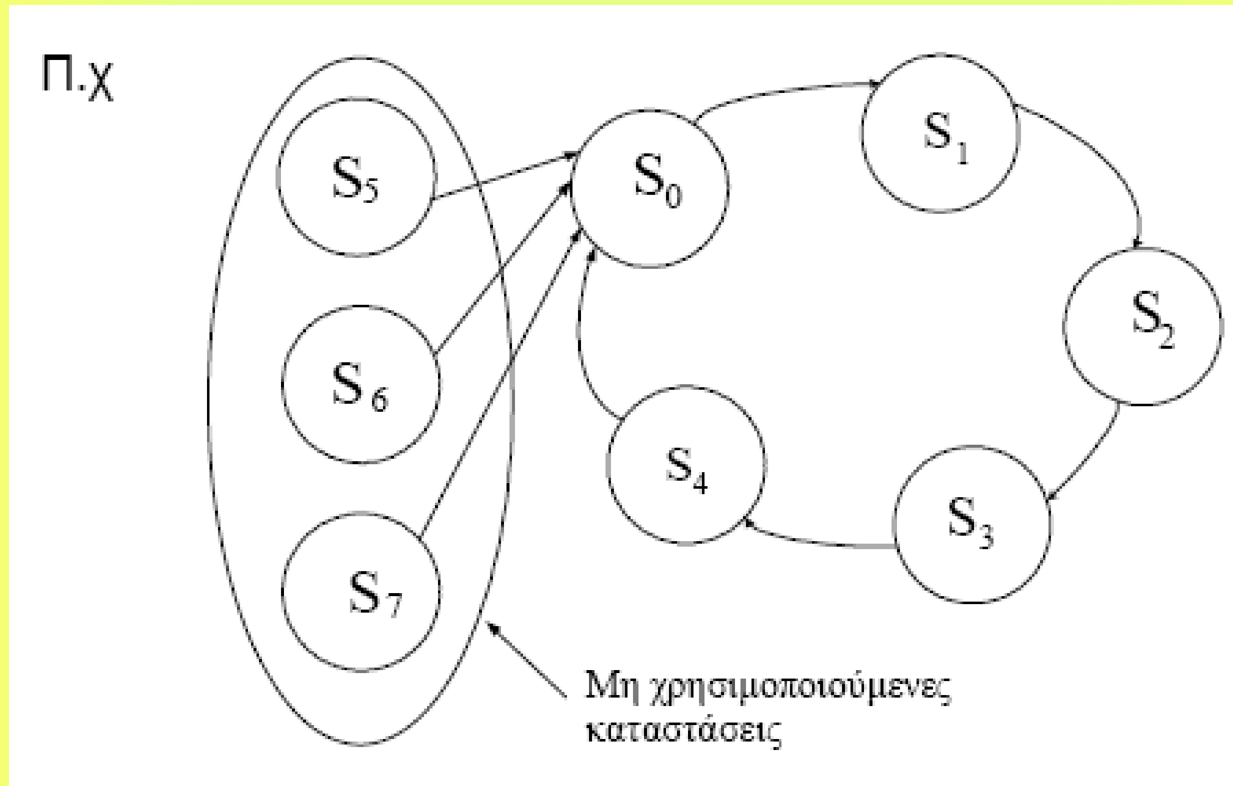
Ένα ακολουθιακό κύκλωμα μπορεί να ξεκινήσει από μια κατάσταση η οποία δεν χρησιμοποιείται, ή να εμφανίσει μια τέτοια κατάσταση λόγω θορύβου. Σ΄ αυτή τη περίπτωση η συμπεριφορά του κυκλώματος είναι μη προβλέψιμη.

Δηλαδή ένα σύστημα μπορεί να έχει το παρακάτω διάγραμμα καταστάσεων



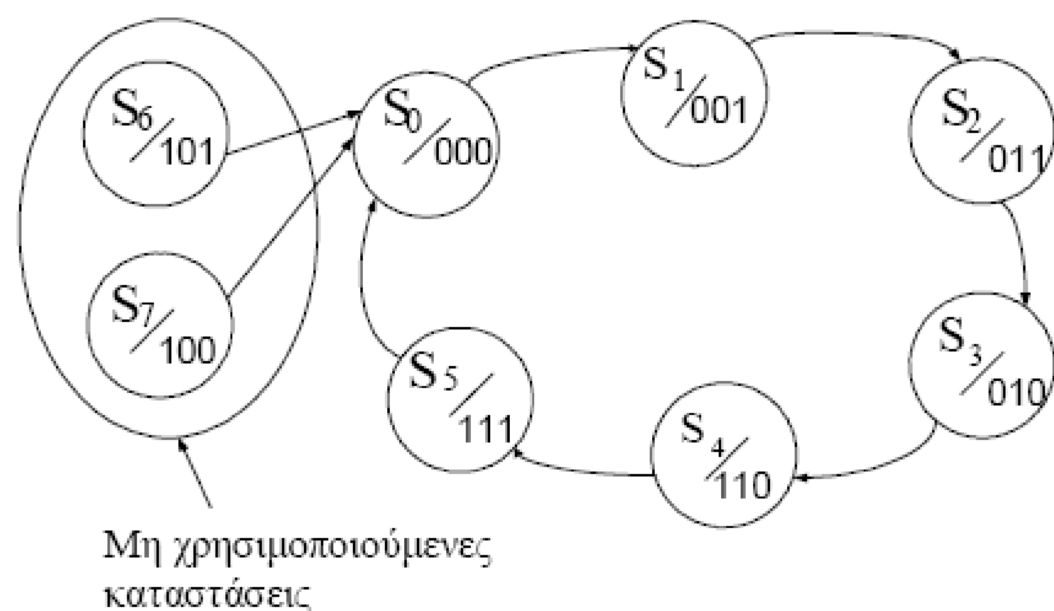
Υπάρχουν τρεις καταστάσεις οι οποίες δεν χρησιμοποιούνται και οι οποίες μπορεί να εμφανίσουν διάφορες αλληλεπιδράσεις που εξαρτώνται από τον τρόπο υλοποίησης του κυκλώματος.

Περιλαμβάνονται στο διάγραμμα καταστάσεων και οι καταστάσεις που δεν χρησιμοποιούνται, σε μια λογική επαναφοράς στην αρχική κατάσταση (Reset circuitry). Δηλαδή κατασκευάζεται ένα διάγραμμα καταστάσεων το οποίο δεν επιτρέπει την εμφάνιση καταστάσεων ‘παγίδα’, ή εάν υπάρξουν, η εμφάνιση τους να έγινε πριν την έναρξη του ρολογιού.



Παράδειγμα

Να σχεδιαστεί ο μετρητής MOD-6 Gray Code χρησιμοποιώντας D-ffs με διάγραμμα καταστάσεων



Θεωρούμε στο παράδειγμα μας ότι οι καταστάσεις PS είναι ίδιες με τις καταστάσεις εξόδου.

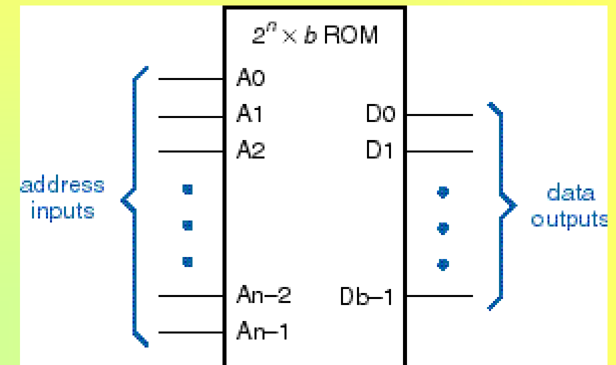
Απάντηση

Διατάξεις ROM

- Η ROM είναι ένα προγραμματιζόμενο ολοκληρωμένο.
- Ο προγραμματισμός μπορεί να γίνει :
  - Μόνο μια φορά :
    - Στο εργοστάσιο (PROM)
    - Από το χρήστη (FPROM)
  - Περισσότερες φορές με σβήσιμο των ήδη υπαρχόντων δεδομένων, με διάφορους τρόπους :
    - Με ακτινοβολία (EPROM)
    - Με ηλεκτρικές διατάξεις (EEPROM – E^2PROM)
  - Τμηματικά
    - Flash
    - Paged flash

Βασικές δομές ROM

- Συνδυαστικό κύκλωμα ν εισόδων και β εξόδων
- Είσοδοι = Διευθύνσεις. Εξοδοι = δεδομένα.
- Η μνήμη ROM "αποθηκεύει" τον πίνακα αληθείας οποιασδήποτε συνάρτησης με εισόδους ≤ είσοδοι της ROM και εξόδους ≤ έξοδοι της ROM.

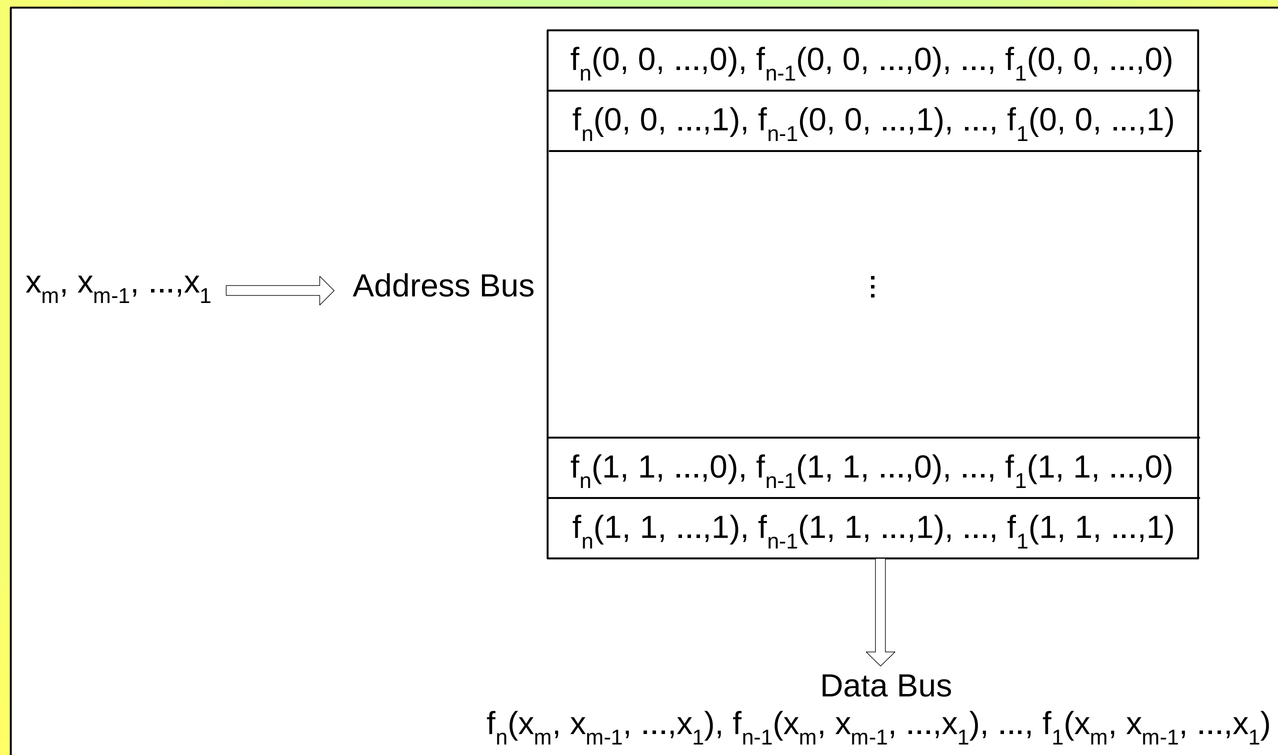


- Ο διπλανός πίνακας αληθείας μπορεί να αποθηκευτεί σε μια ROM (8x4)
- Η ROM τελικά είναι μνήμη ή συνδυαστικό κύκλωμα ?
- Η ROM είναι nonvolatile !

Inputs			Outputs			
A2	A1	A0	D3	D2	D1	D0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Υλοποίηση συναρτήσεων με μνήμη

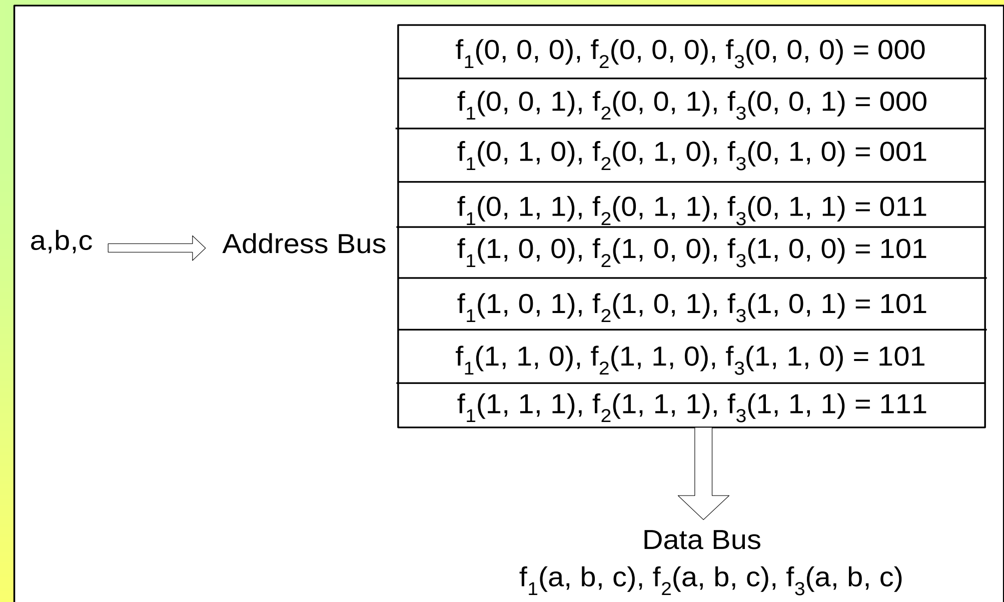
- $f_1(x_1, x_2, x_3, ..., x_m)$, $f_2(x_1, x_2, x_3, ..., x_m)$, …, $f_n(x_1, x_2, x_3, ..., x_m)$
- Η υπόθεση ότι κάθε συνάρτηση είναι των ίδιων μεταβλητών είναι απόλυτα λογική καθώς οποιαδήποτε λιγότερων μπορεί να μετατραπεί σε συνάρτηση m μεταβλητών, όπου οι επιπλέον είναι αδιάφοροι.
- $f(a,b,c) = ab + c$
- $f'(a,b,c,d) = (ab+c)1 = (ab+c)(d + !d) = abd + cd + ab!d + c!d$



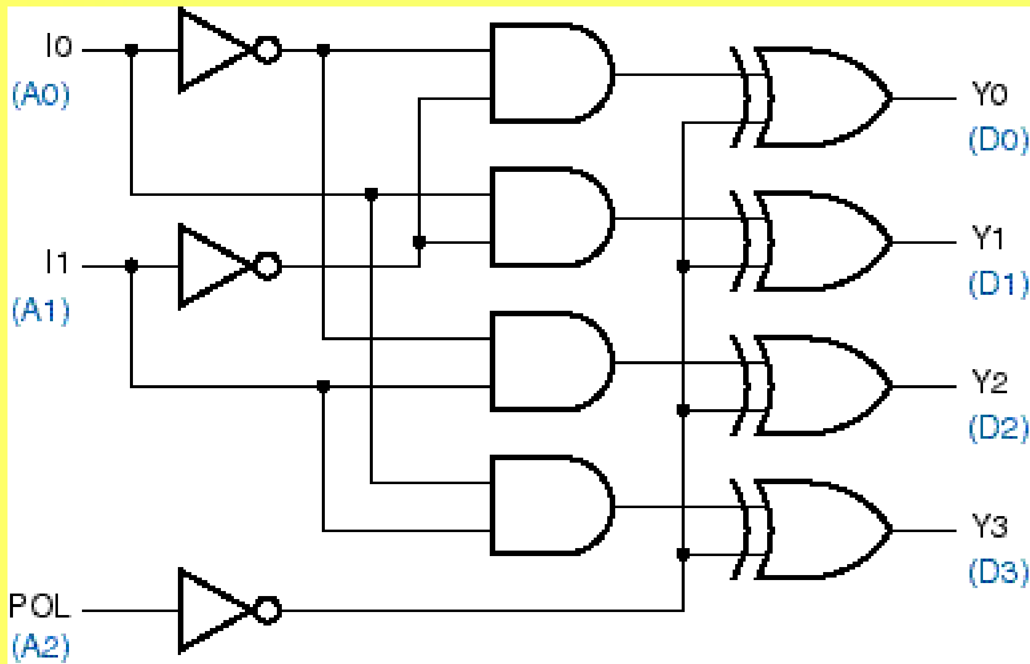
Παράδειγμα υλοποίησης συναρτήσεων με μνήμη

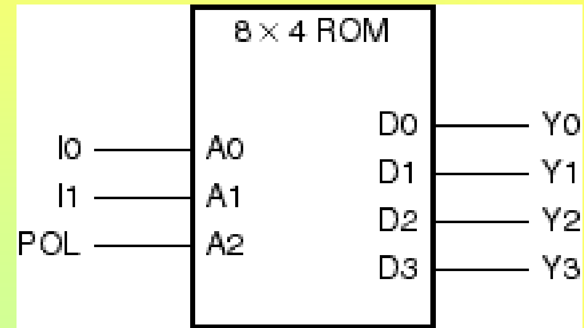
- $f_1(a) = a$, $f_2(b, c) = bc$, $f_3(a, b) = a + b$
- Αριθμός μεταβλητών m = 3
- $f'_1(a, b, c) = abc + ab!c + a!bc + a!b!c$,
- $f'_2(a, b, c) = abc + !abc$
- $f'_3(a, b, c) = abc + ab!c + a!bc + a!b!c + !abc + !ab!c$

a	b	c	f'_1	f'_2	f'_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	1



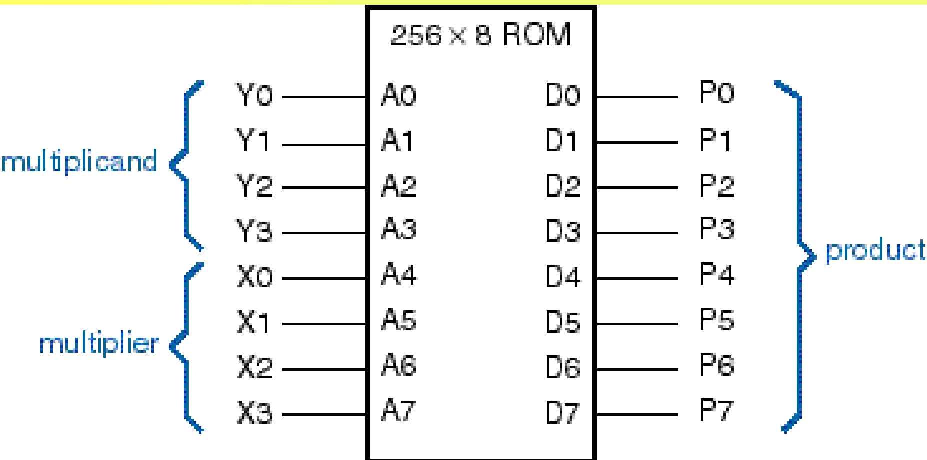
Οι δύο υλοποιήσεις ενός συνδυαστικού κυκλώματος





- Οι δύο υλοποιήσεις είναι ισοδύναμες μόνο από λογικής πλευράς.
- Από χρονικής πλευράς η ROM είναι αφενώς πιο αργή, αφετέρου παρέχει τον ίδιο χρόνο απόκρισης ανεξάρτητα των εισόδων POL, I1 και I0.
- Για κύκλωμα 4 μεταβλητών χρειαζόμαστε ROM 16 x Y. Για ένα 20 μεταβλητών 1M x Y !!!
- Υλοποίηση με ROM συμφέρει :
  - Για μικρά κυκλώματα λίγων εισόδων
  - Για να κρύψουμε τη πραγματική υλοποίηση.

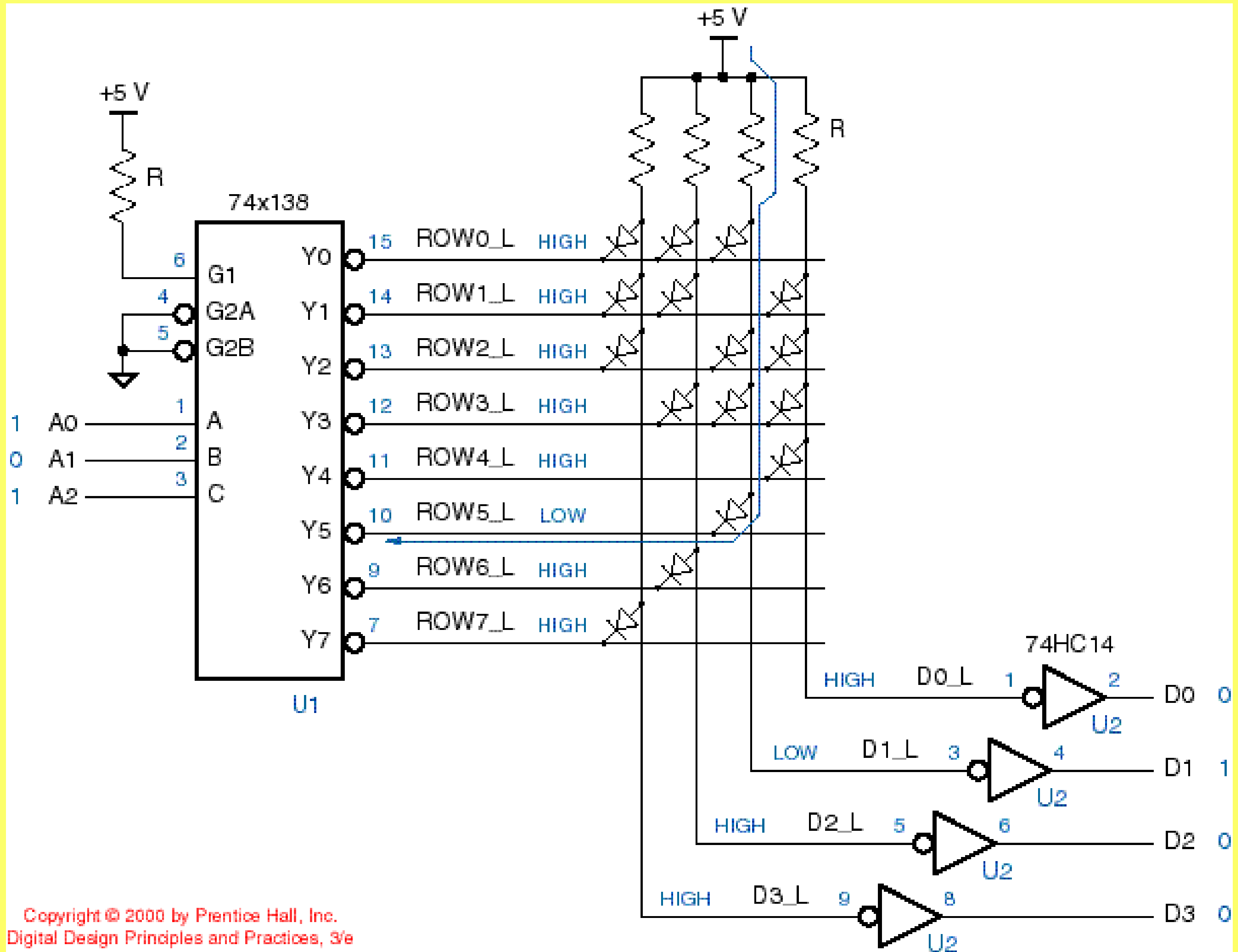
Πολλαπλασιασμός με ROM !



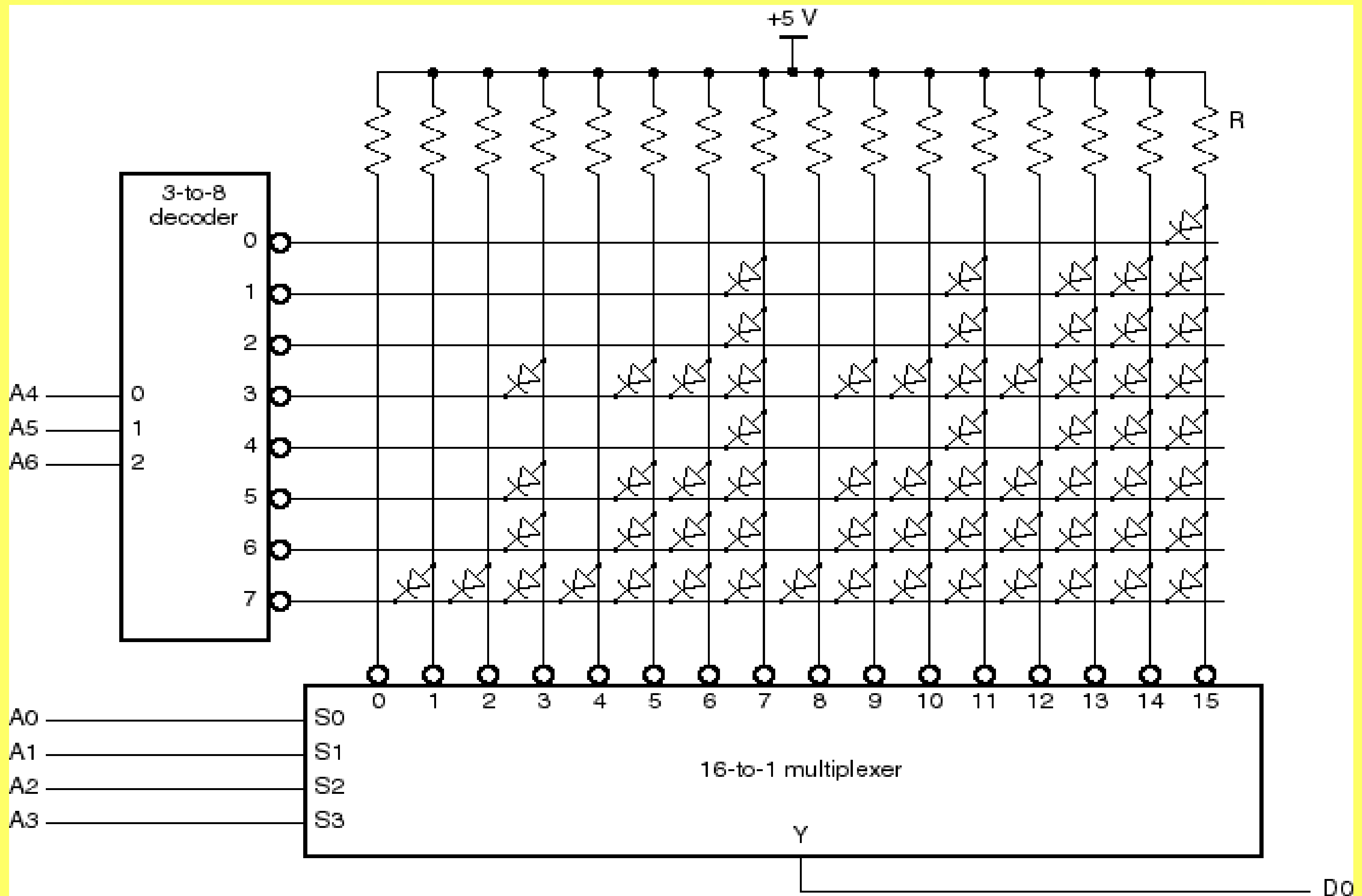
00:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
20:	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
30:	00	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
40:	00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
50:	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
60:	00	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
70:	00	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
80:	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
90:	00	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A0:	00	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B0:	00	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C0:	00	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D0:	00	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E0:	00	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F0:	00	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

- Για τον πολλαπλασιασμό 2 αριθμών των 4 δ.ψ. θα μπορούσα να χρησιμοποιήσω μια ROM 256x8.
- Η διεύθυνση σχηματίζεται από τη συνένωση του πολλαπλασιαστή και του πολλαπλασιαστέου.
- Το αποτέλεσμα των 8 δ.ψ. είναι η έξοδος δεδομένων της ROM.
- Στη θέση {Y, X} στη ROM θα πρέπει να έχουμε αποθηκευμένη τη τιμή Y x X.
- Ο προγραμματισμός γίνεται με ειδική συσκευή στην οποία δίνουμε τα δεδομένα με τη μορφή πίνακα.

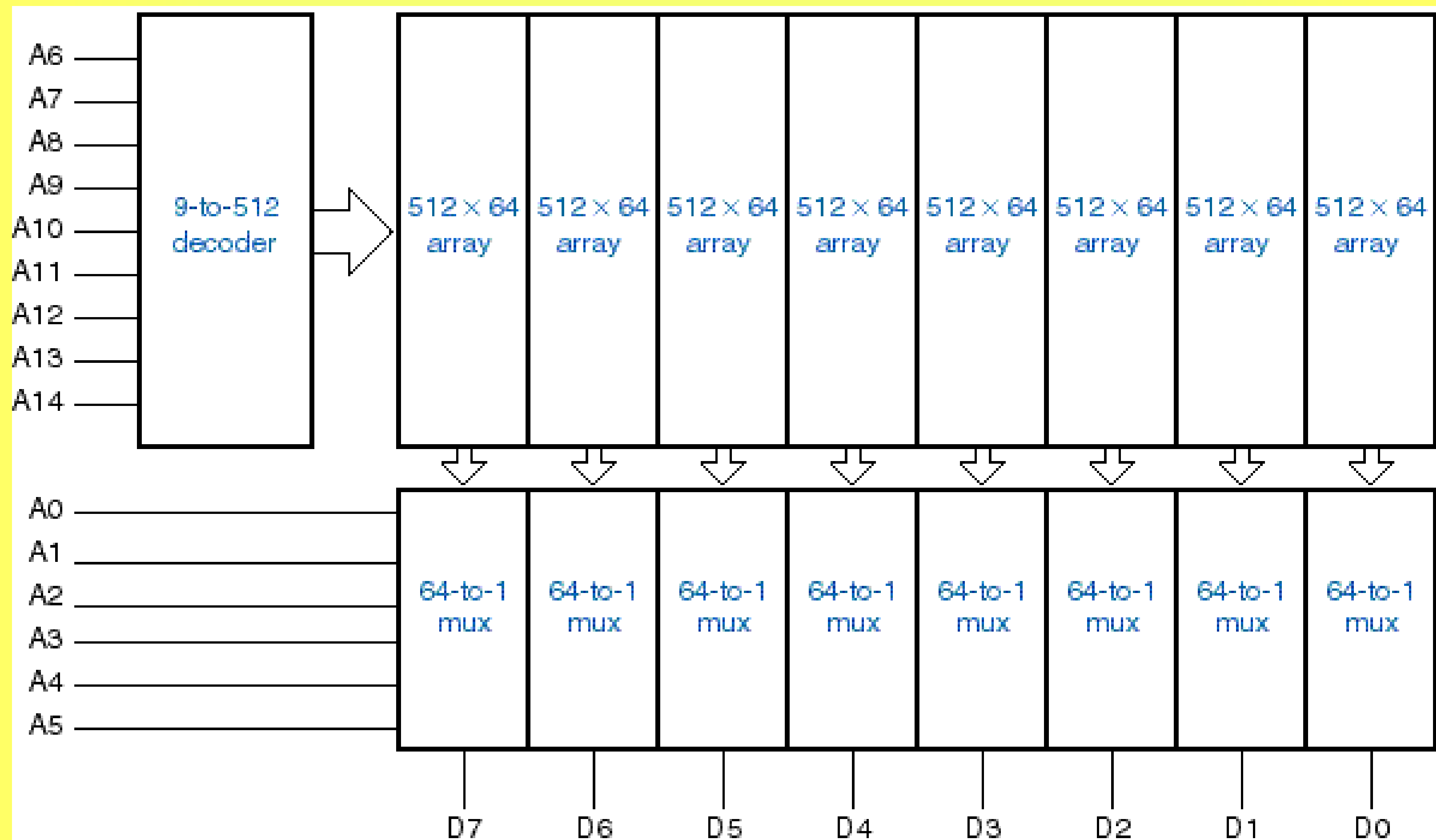
Εσωτερικά σε μια ROM



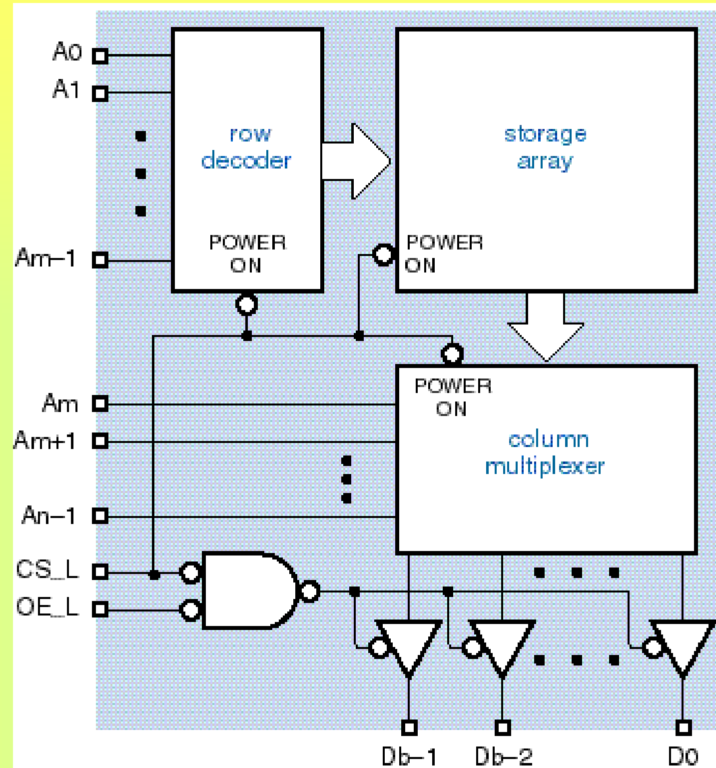
Δισδιάστατη αποκωδικοποίηση : 128x1



Πιθανή διάταξη μιας 32K x 8

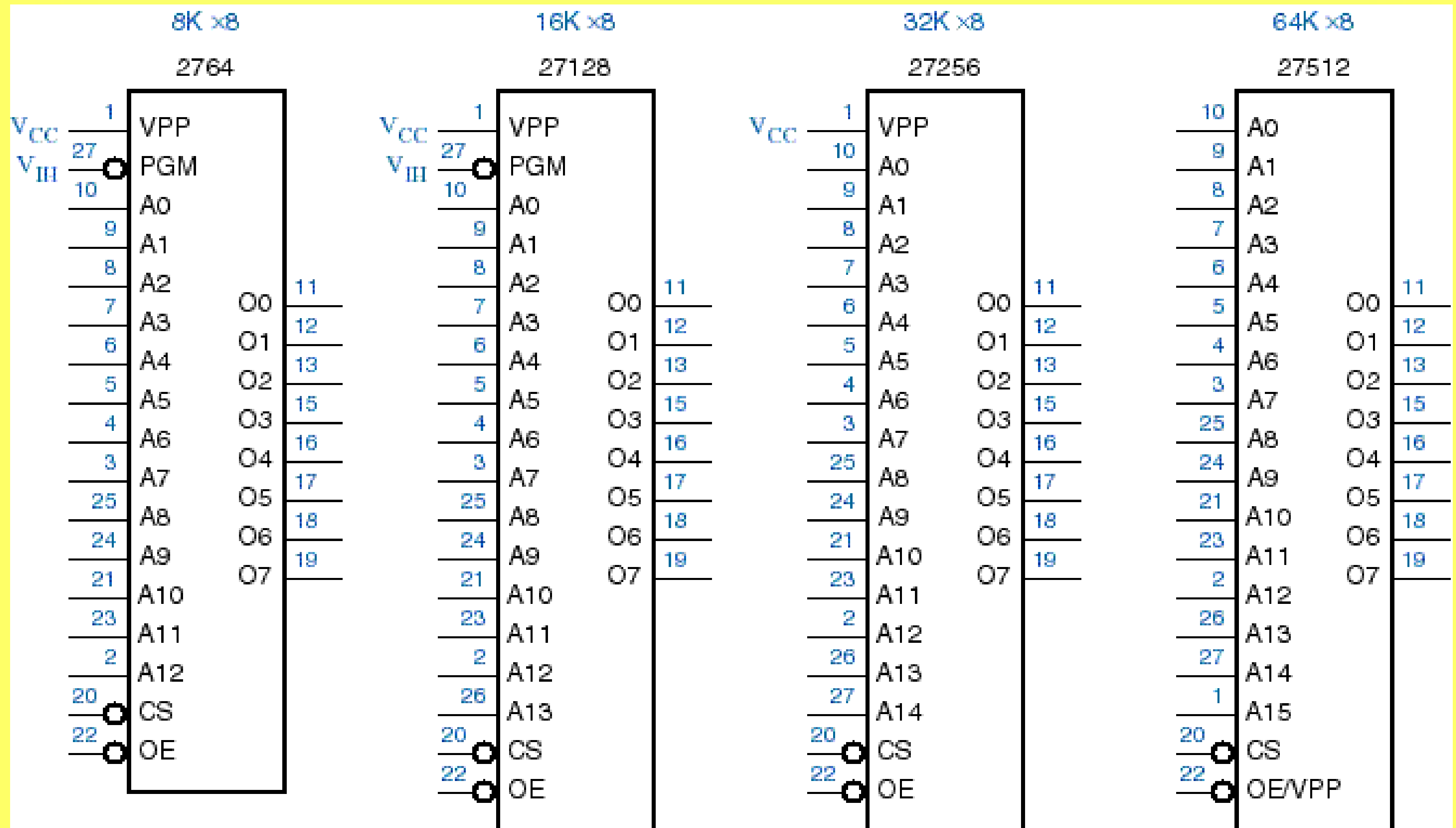


Γενική αρχιτεκτονική και κρίσιμοι χρόνοι μιας ROM



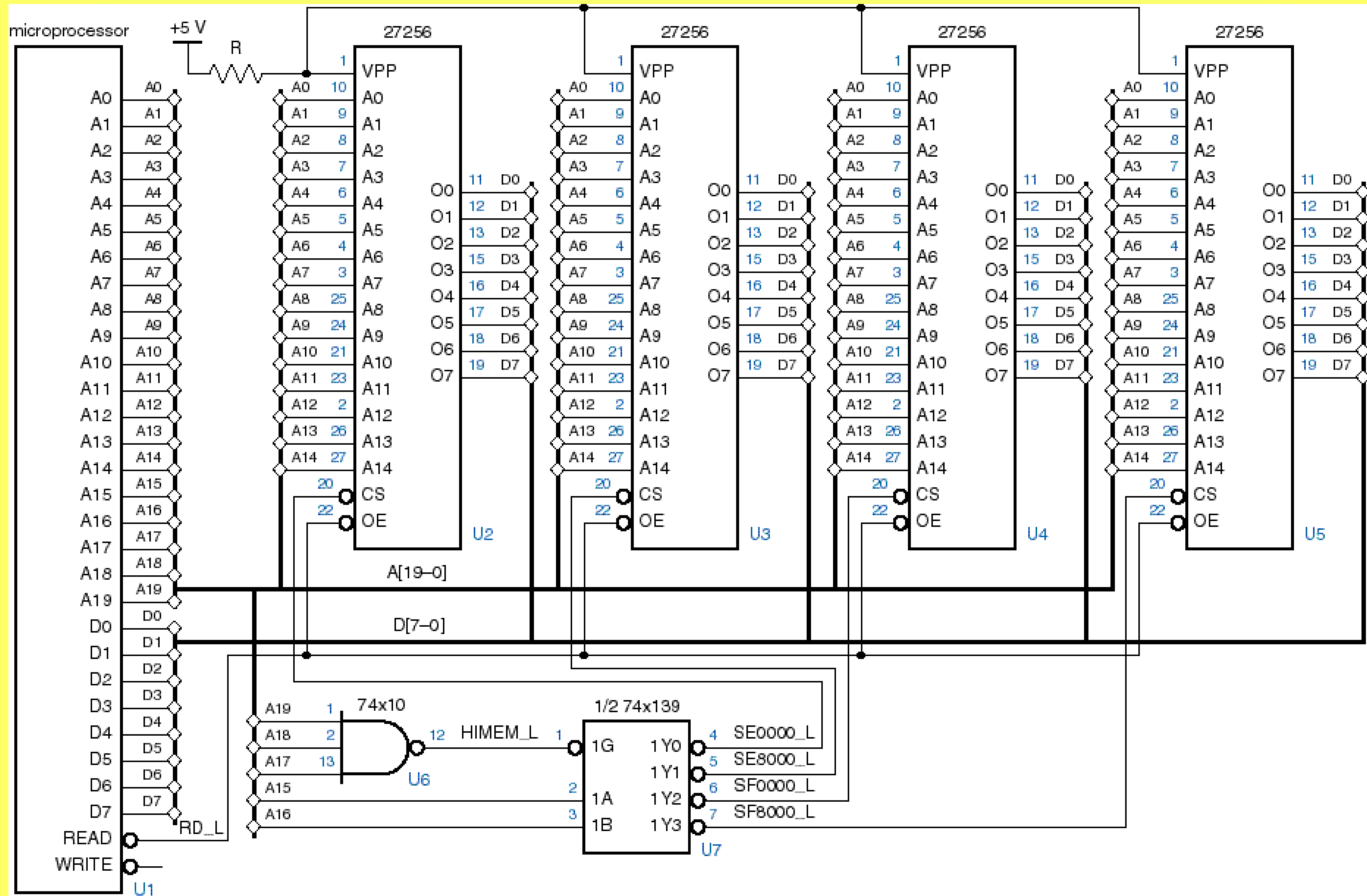
ADDR
stable
stable
stable
$\geq t_{AA}$
$\max(t_{AA}, t_{ACS})$
CS_L
t_{ACS}
t_{OH}
OE_L
t_{AA}
t_{OZ}
t_{OE}
t_{OZ}
t_{OE}
DATA
valid
valid
valid

LSI EPROMs



8K x8
2764
VCC
VIH
1 VPP
27 PGM
10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
20 CS
22 OE
O0 11
O1 12
O2 13
O3 15
O4 16
O5 17
O6 18
O7 19
16K x8
27128
VCC
VIH
1 VPP
27 PGM
10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
26 A13
20 CS
22 OE
O0 11
O1 12
O2 13
O3 15
O4 16
O5 17
O6 18
O7 19
32K x8
27256
VCC
1 VPP
10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
26 A13
27 A14
20 CS
22 OE
O0 11
O1 12
O2 13
O3 15
O4 16
O5 17
O6 18
O7 19
64K x8
27512
10 A0
9 A1
8 A2
7 A3
6 A4
5 A5
4 A6
3 A7
25 A8
24 A9
21 A10
23 A11
2 A12
26 A13
27 A14
1 A15
20 CS
22 OE/VPP
O0 11
O1 12
O2 13
O3 15
O4 16
O5 17
O6 18
O7 19

Φτιάχνοντας μεγαλύτερες ROMs από μικρότερες



- ΤΕΛΟΣ ΨΗΦΙΑΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ

Γλώσσες περιγραφής Υλικού
Verilog, VHDL

Hardware Description Languages (HDLs)
Γλώσσες περιγραφής υλικού

- **Γιατί υπάρχουν ?**
  - Οι γλώσσες προγραμματισμού δε μας καλύπτουν
  - Το υλικό έχει μια ενγενή παραλληλία
  - Στις γλώσσες που ξέρουμε υποθέτουμε εκτέλεση σε μια ακολουθιακή μηχανή.
  - Η εκτέλεση στο υλικό δεν είναι ακολουθιακής φύσης.

- **Γιατί χρειάζονται ?**
  - Μεταφερσιμότητα μεταξύ τεχνολογιών
  - Κοινό υπόβαθρο συνεννόησης μεταξύ σχεδιαστών.
  - Εύκολη μετάβαση από περιγραφή σε υλοποίηση

Γλώσσες περιγραφής υλικού

- Τι επιπλέον προσφέρουν ?
  - Αναπαραστάσεις σε διάφορες μορφές που χρησιμοποιεί ένας σχεδιαστής υλικού : λογικά διαγράμματα, συναρτήσεις Boole, FSMs …
  - Εξομοίωση (Simulation)
  - Σύνθεση (Synthesis)

Εξομοίωση

- **Λογική Εξομοίωση (Logic Simulation)**
  - Πιστοποίηση της σωστής δομής και συμπεριφοράς ενός κυκλώματος με τη χρήση υπολογιστή και κατάλληλου s/w, *πριν τη πραγματική του υλοποίηση*.
  - Simulator (Εξομοιωτής)
  - Είσοδοι : 1) περιγραφή κυκλώματος σε HDL
2) διανύσματα εισόδου (stimuli)
  - Stimulus file (test bench)
  - Εξοδος : Τιμές εξόδων του κυκλώματος.

- Σε τι είναι γραμμένο ένα stimulus file ?
  - Φυσικά, σε HDL !!!

Σύνθεση

Logic synthesis

- Αυτοματοποιημένη διαδικασία παραγωγής του δικτυώματος (netlist) ενός κυκλώματος που έχουμε περιγράψει σε HDL, για κάποια συγκεκριμένη τεχνολογία. Το netlist είναι η λίστα των σχεδιαστικών κυττάρων που χρησιμοποιούμε καθώς και η διασύνδεσή τους για τη στοχευόμενη λειτουργικότητα.

- Synthesizer / Synthesis tool

- Είσοδοι : 1) περιγραφή κυκλώματος σε υποσύνολο της HDL
  2) επιθυμητά χαρακτηριστικά λειτουργίας.

- Εξοδος : Δικτύωμα του κυκλώματος.

Παραλληλία στις HDLs

- Κλασσικές γλώσσες προγραμματισμού :
  - A=B; C=A; => C=B
  - C=A; A=B; => C= A
  - Η σειρά αναγραφής των εντολών ΕΧΕΙ ΣΗΜΑΣΙΑ γιατί ο προγραμματιστής λαμβάνει υπόψη του το ακολουθιακό μοντέλο εκτέλεσης

- Αρχή της παραλληλίας στις HDLs :
  - A=B; C=A; => C=B
  - C=A; A=B; => C=B
  - Η σειρά αναγραφής των εντολών ΔΕΝ ΕΧΕΙ ΣΗΜΑΣΙΑ γιατί περιγράφουμε H/W που εκ φύσεως είναι παράλληλο.

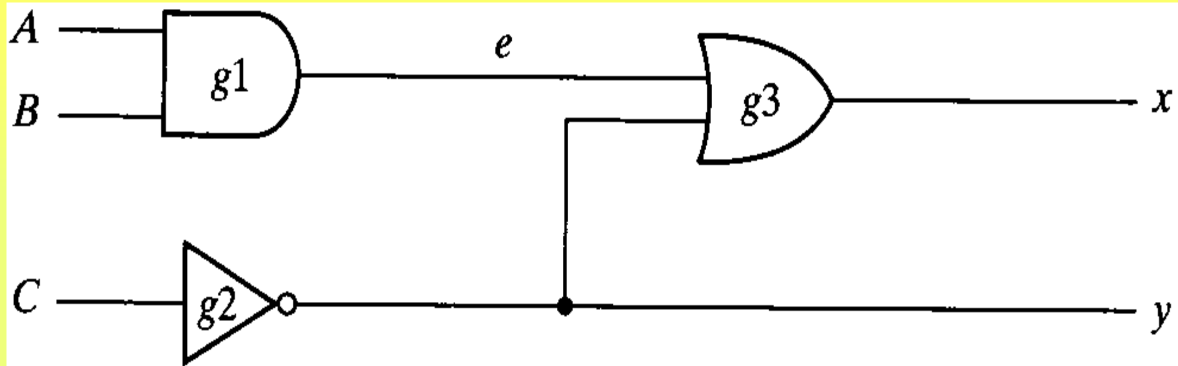
Verilog

- C like. Λιγότερες δηλώσεις, λιγότερος κώδικας, αντίστοιχο functionality.
- Πρωτοεισήχθηκε από την Gateway Design System Corp. 1985 (μετέπειτα Cadence)
- IEEE standard 1985.
- Open Verilog International (OVI).
- Προτεινόμενο σύγγραμμα : "The Verilog Hardware Description Language", 3rd Ed., D. E. Thomas & P. R. Moorby, KAP, 1996.
- Χρησιμοποιεί περίπου 100 keywords, π.χ. and, or, module …
- Case sensitive, τα κενά αγνοούνται.
- Οτιδήποτε ακολουθεί το // είναι comment.

Verilog

- Η περιγραφή ενός κυκλώματος σε Verilog, μπορεί να :
  - είναι πλήρως δομική (structural description)
  - βασίζεται στη συμπεριφορά του κυκλώματος (behavioral)
  - συνδυάζει σε οποιονδήποτε βαθμό τα δύο παραπάνω.
- Ενας σχεδιασμός όσο μεγάλος ή μικρός είναι στη Verilog αποτελεί ένα module.
- Η περιγραφή ενός module μπορεί να χτίζεται με τη χρήση αντιγράφων άλλων modules.
- Δημιουργείται έτσι μια σχεδιαστική ιεραρχία με το πλέον υψηλό να είναι το top level module.

Το πρώτο παράδειγμα




```
module cir1 (A, B, C, x, y);
        input A, B, C;
        output x, y ;
        wire e;
        and g1 (e, A, B);
        not g2 (y, C);
        or g3 (x, e, y);
endmodule
```

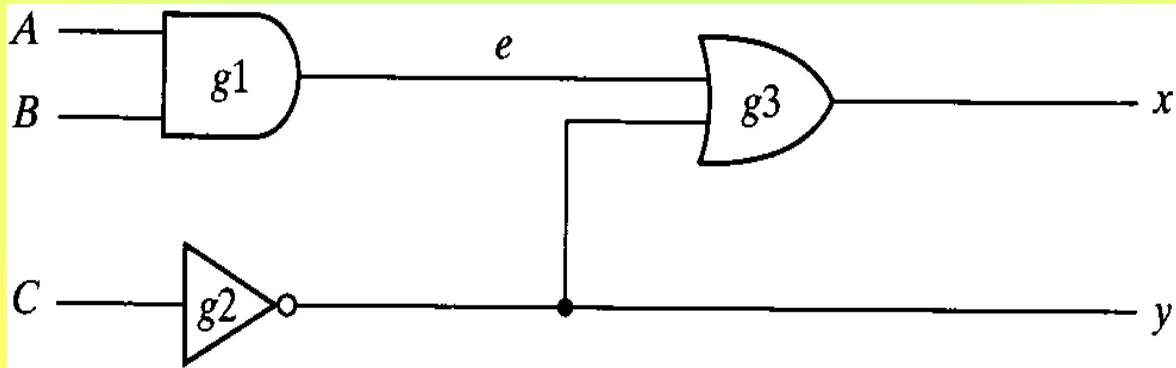
Η πρώτη μεταβλητή είναι πάντοτε η έξοδος

Predefined (primitive) modules

Πρόσθεση Χρονοκαθυστερήσεων

```
module cir2 (A, B, C, x, y);
        input A, B, C;
        output x, y ;
        wire e;
        and #30 g1 (e, A, B);
        not #10 g2 (y, C);
        or # 20 g3 (x, e, y);
endmodule
```

X = χρονοκαθυστέρηση X χρονικών μονάδων εξομοίωσης

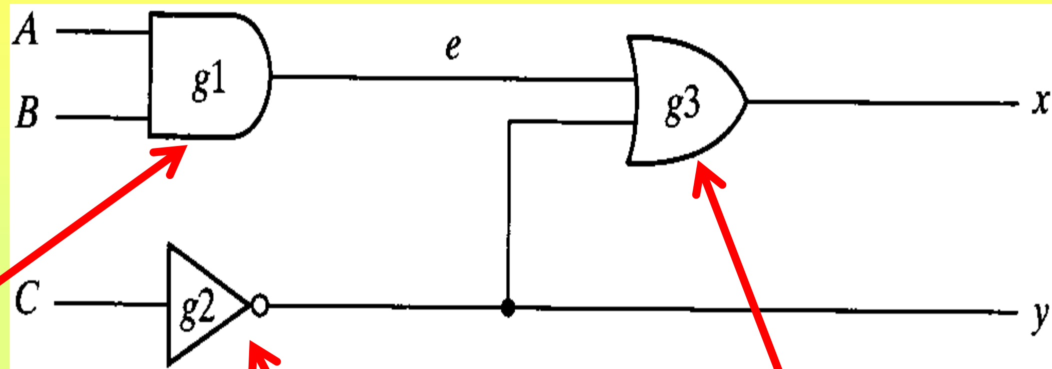


Testbench & Simulation

```
module testbench;
   reg K, L, M;
   wire my_x, my_y ;
cir2 CUT (K, L, M, my_x, my_y);
initial
  begin
          K=0; L=0; M=0;
#100      K=1; L=1; M=1;
#100      $stop;
  end
endmodule
```

```
module cir2 (A, B, C, x, y);
          input A, B, C;
          output x, y ;
          wire e;
          and #30 g1 (e, A, B);
          not #10 g2 (y, C);
          or # 20 g3 (x, e, y);
endmodule
```

Testbench & Simulation



30 μονάδες καθυστέρηση

20 μονάδες καθυστέρηση

10 μονάδες καθυστέρηση

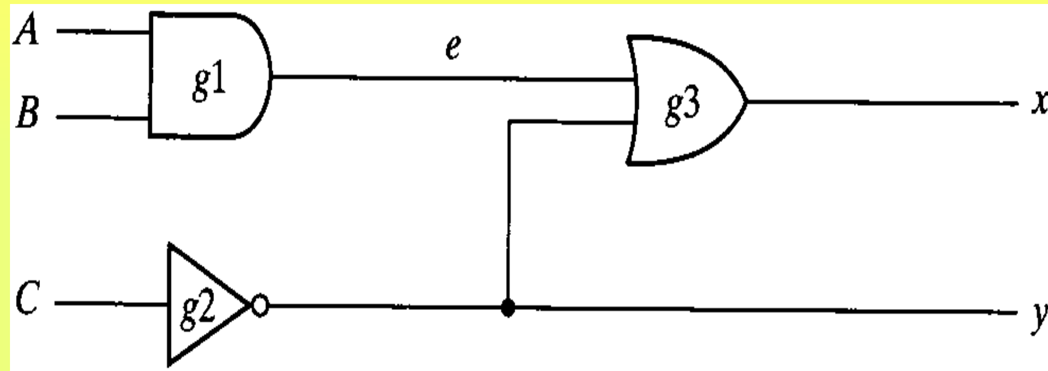
/A = St1
/B = St1
/C = St1
/x = St1
/y = St0
/e = St1

0 50 100 150

Equation Description – Συνδυαστικά Κυκλώματα

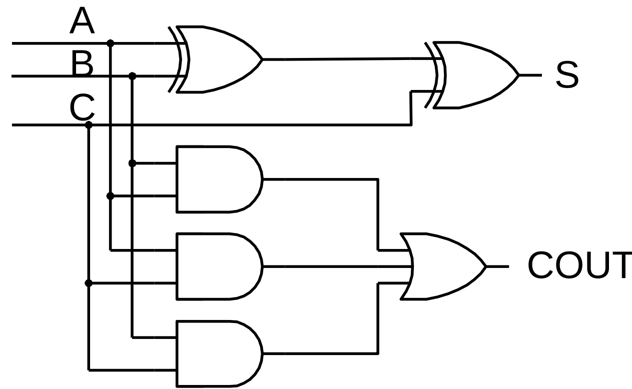
- Αντί να περιγράφουμε τη δομή του κυκλώματος, περιγράφουμε τη λογική συμπεριφορά του, χρησιμοποιώντας συναρτήσεις Boole, πίνακες αληθείας …
- Αν και μπορούμε να χρησιμοποιήσουμε τις predefined συναρτήσεις AND, OR, XOR … συνήθως χρησιμοποιούμε τις συντομογραφίες τους :
  - & = AND
  - | = OR
  - ~ = NOT / Complement
  - ^ = XOR
  - ~^ = XNOR

Equation Description




```
module cir2 (A, B, C, x, y);
        input A, B, C;
        output x, y ;

        assign y = ~C;
        assign x = (A & B) | y;
endmodule
```

Για τον ίδιο σχεδιασμό υπάρχουν :



Δομικές Περιγραφές

```
module OneBitFullAdder(A,B,C,S,COUT);
 input A,B,C;
 output S, COUT ;
 wire HalfSum,O1,O2,O3;

 xor  g1 (HalfSum,A,B);
 xor  g2 (S,HalfSum,C);
 and g3 (O1,A,B);
 and g4 (O2,A,C);
 and g5 (O3,B,C);
  or  g6 (COUT,O1,O2,O3);
endmodule
```

Περιγραφή Εξισώσεων

```
module OneBitFullAdder(A,B,C,S,COUT);
 input A, B, C;
 output S, COUT ;

 assign S= A^B^C;
 assign COUT= (A&B) | (A&C) | (B&C);
endmodule
```

Ποια περιγραφή είναι καλύτερη ?

Δομικές περιγραφές

- Περιγράφουν μία μόνο υλοποίηση με απόλυτη ακρίβεια
- Χρειάζονται πολύ περισσότερη ώρα εξομοίωσης

Υψηλού επιπέδου περιγραφές

- Δε περιγράφουν το κύκλωμα με απόλυτη ακρίβεια
- Περιγράφουν όλα τα ισοδύναμα κυκλώματα
- Εξομοιώνονται πολύ πιο γρήγορα

Παράδειγμα εναλλακτικών περιγραφών & ιεραρχίας (1/5)

– Σχεδιάστε ένα κύκλωμα που δέχεται στην είσοδό του έναν φυσικό αριθμό A των 4 δυαδικών ψηφίων και παράγει στην έξοδό του 1 αν ο αριθμός είναι πολλαπλάσιο του 3.

– Ομοίως για πολλαπλάσιο του 5.

– Ομοίως για πολλαπλάσιο του 15, χρησιμοποιώντας μόνο τα κυκλώματα των προηγούμενων υποερωτημάτων.

Μπορούμε να φτιάξουμε πίνακα αλήθειας για το 1ο υποερώτημα, απ' όπου προφανώς θα πάρουμε ότι t(A) = t(A[3], A[2], A[1], A[0]) = Σ(0, 3, 6, 9, 12, 15).

A[3] A[2] \ A[1] A[0]	00	01	11	10
00	1		1	
01				1
11	1		1	
10		1		

$$t(A[3], A[2], A[1], A[0]) =$$
$$A'[3]\,A'[2]\,A'[1]\,A'[0] + A'[3]\,A'[2]\,A[1]\,A[0] + A'[3]\,A[2]\,A[1]\,A'[0] +$$
$$A[3]\,A[2]\,A'[1]\,A'[0] + A[3]\,A[2]\,A[1]\,A[0] + A[3]\,A'[2]\,A'[1]\,A[0] =$$
$$A'[3]\,A'[2]\,(A[1] \odot A[0]) + A[3]\,A[2]\,(A[1] \odot A[0]) +$$
$$A'[3]\,A[2]\,A[1]\,A'[0] + A[3]\,A'[2]\,A'[1]\,A[0] =$$
$$(A[3] \odot A[2])\,(A[1] \odot A[0]) + A'[3]\,A[2]\,A[1]\,A'[0] + A[3]\,A'[2]\,A'[1]\,A[0]$$

Παράδειγμα εναλλακτικών περιγραφών & ιεραρχίας (2/5)

A[3] A[2] \ A[1] A[0]	00	01	11	10
00	1		1	
01				1
11	1		1	
10		1		

$$t(A[3], A[2], A[1], A[0]) =$$
$$A'[3]\,A'[2]\,A'[1]\,A'[0] + A'[3]\,A'[2]\,A[1]\,A[0] + A'[3]\,A[2]\,A[1]\,A'[0] +$$
$$A[3]\,A[2]\,A'[1]\,A'[0] + A[3]\,A[2]\,A[1]\,A[0] + A[3]\,A'[2]\,A'[1]\,A[0] =$$
$$A'[3]\,A'[2]\,(A[1] \odot A[0]) + A[3]\,A[2]\,(A[1] \odot A[0]) +$$
$$A'[3]\,A[2]\,A[1]\,A'[0] + A[3]\,A'[2]\,A'[1]\,A[0] =$$
$$(A[3] \odot A[2])\,(A[1] \odot A[0]) + A'[3]\,A[2]\,A[1]\,A'[0] + A[3]\,A'[2]\,A'[1]\,A[0]$$

Πλέον μπορούμε πριν κατασκευάσουμε το κύκλωμά μας να το περιγράψουμε :

Δομικά

```
module Multiple3 (A, t);
   input [3:0] A;
   output t;

   xnor g0 (temp1, A[3], A[2]);
   xnor g1 (temp2, A[1], A[0]);
   and  g2 (temp3, temp1, temp2);
   not  g3 (nota3, A[3]);
   not  g4 (nota2, A[2]);
   not  g5 (nota1, A[1]);
   not  g6 (nota0, A[0]);
   and  g7 (temp4, nota3, A[2], A[1], nota0);
   and  g8 (temp5, A[3], nota2, nota1, A[0]);
   or   g9 (t, temp3, temp4, temp5);
endmodule
```

Με τις λογικές του εξισώσεις :

```
module Multiple3 (A, t);
   input [3:0] A;
   output t;

   assign temp3 = (A[3]~^A[2]) & (A[1]~^A[0]);
   assign temp4 = ~A[3] & A[2] & A[1] & ~A[0];
   assign temp5 = A[3] & ~A[2] & ~A[1] & A[0];
   assign t = |{temp3, temp4, temp5};
endmodule
```

Παράδειγμα εναλλακτικών περιγραφών & ιεραρχίας (3/5)

– Ναι, αλλά έτσι έχω κάνει εγώ το περισσότερο κόπο !
– Μήπως υπάρχει κάποιος τρόπος, να μην ασχοληθώ καθόλου με εξισώσεις, απλοποίηση και υλοποίηση ?
– Με άλλα λόγια υπάρχει ένα ακόμα πιο υψηλό επίπεδο αφαίρεσης που να μου δίνει άμεσα περιγραφές, εξομοιώσιμες μα κυρίως συνθέσιμες ?
– Φυσικά και υπάρχει κι αυτός είναι ο λόγος για τον οποίο φτιάξαμε εξ αρχής μια νέα γλώσσα.
– Είναι η περιγραφή βάσει της συμπεριφοράς (behavioral description).

Βάσει συμπεριφοράς :

```
module Multiple3 (A, t);
   input [3:0] A;
   output t;

   assign t = ((A%3) == 0) ? 1 : 0;
endmodule
```

Παράδειγμα εναλλακτικών περιγραφών & ιεραρχίας (4/5)

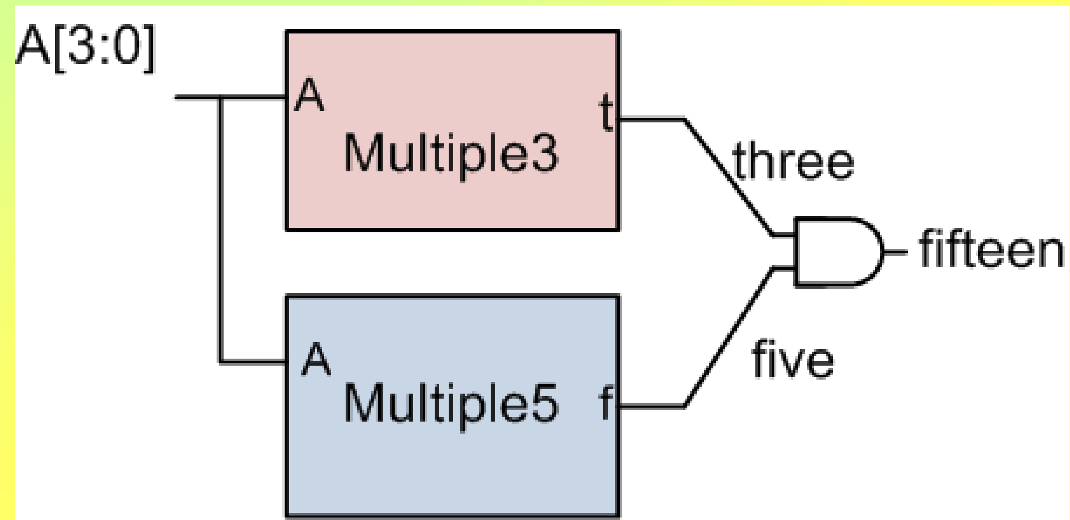
Βάσει συμπεριφοράς 2

```
module multiple3 (A, t);
  input  [3:0] A ;
  output t;
  reg t;
  always @(A)
    begin
      case (A)
        0, 3, 6, 9, 12, 15 : t = 1;
        default : t = 0;
      endcase
    end
endmodule
```

Παράδειγμα εναλλακτικών περιγραφών & ιεραρχίας (5/5)

- Ας δούμε τώρα το 3ο ερώτημα, αφού το 2ο το αντιμετωπίζουμε όπως το 1ο.
- Οπως χρησιμοποίησα τις xor, and, or, … για να χτίσω ένα σχεδιασμό, μπορώ να χρησιμοποιήσω το σχεδιασμό σα κύτταρο για τη περιγραφή (δομική) μεγαλύτερων σχεδιασμών.
- Σκεφτείτε για παράδειγμα ότι έχω φτιάξει 2 υποσχεδιασμούς το multiple5 και το multiple3 που μου δείχνουν στις εξόδους f και t αντίστοιχα αν η είσοδος A[3:0] είναι πολλαπλάσιο του 5 και του 3 αντίστοιχα.
- Θεωρείστε τις δηλώσεις
  - module multiple5 (A, f);
  - module multiple3(A,t);
- Για να φτιάξω ένα κύκλωμα που θα μου δείχνει αν η είσοδος A είναι πολλαπλάσιο του 15, αρκεί να τροφοδοτήσω αυτή την είσοδο και στα 2 προηγούμενα modules και να πάρω το λογικό AND των εξόδων τους.
- Θα μπορούσα συνεπώς να γράψω :

```
module multiple15 (A, fifteen);
  input [7:0] A;
  output fifteen;
  wire five, three;

  multiple5 U1 (A, five);
  multiple3 U2 (A, three);
  assign fifteen = five & three;
endmodule
```



Operators στη behavioral περιγραφή

- Οι βασικές λογικές : &, |, ^, ~, ~^
- Οι βασικές αριθμητικές : +, -, *, /, %
- Οι βασικές σύγκρισης : < , >, !=, ==
- Concatenation : { , }
- Conditional / ternary operator (3 – way assignment) (? :)

  condition ? true expression : false expression

```
module magcomp (A,B,ALSB,
                AGTB, AEQB);
  input [3:0] A, B;
  output ALSB, AGTB, AEQB;
      assign      ALSB = (A< B),
                  AGTB = (A>B),
                  AEQB = (A ==B);
endmodule
```

Literals and constants

- Εστω μια αρτηρία : wire [5:0] A;
- Τι σημαίνει η δήλωση A = 3 ? Σημαίνει A = 000011 ? Μήπως σημαίνει 11XXXX ? Μήπως A = 110000?
- H Verilog θα υποθέσει το πρώτο !
- Καλό είναι ο προγραμματιστής να ξεκαθαρίζει τι ακριβώς θέλει :
- Literals : n'Fddd….
  - n = αριθμός δυαδικών ψηφίων
  - F = σύστημα αναπαράστασης. b (binary) / o (octal) / d (decimal) / h (hex). Default is decimal !
  - ddd… ψηφία του συστήματος αναπαράστασης.
- Π.χ. assign A[5:0] = 6'b000011, A[5:0] = 6'd3, A[5:0] = 6'h03;

- Πως θα βάλω τη τιμή 01ZXX1 στο A ?

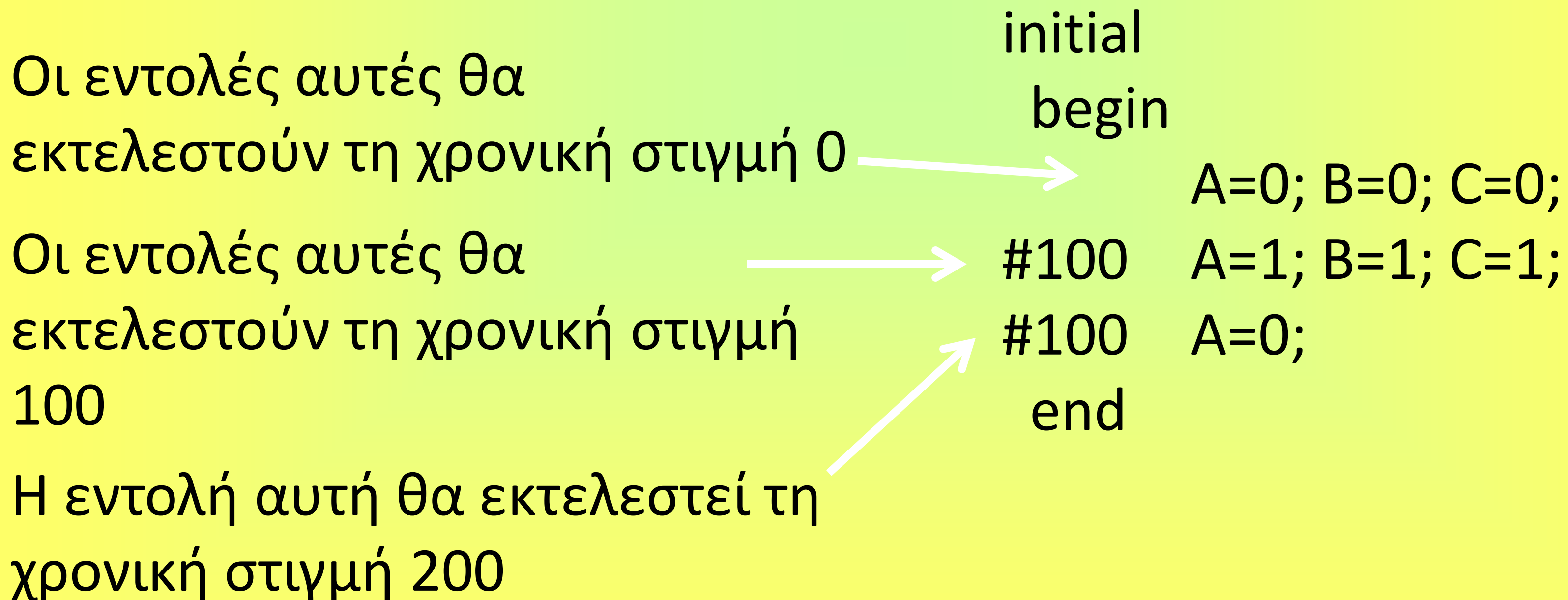
Testbench

```
module test_multiple15 ;
  reg [3:0] in ;
  wire mult ;

  multiple15 p0 (in, mult) ;
  initial begin
     in = 0 ;
     repeat (16) begin
        #100 $display("in = %2d mult = %1b",in,mult) ;
        in = in+1 ;
     end
   end
endmodule
```

- Οταν γράφω ένα testbench έχω πολύ μεγαλύτερη ελευθερία στη χρήση δομών της Verilog.
- Ο κώδικας που γράφω δεν αντιπροσωπεύει κύκλωμα !
- Ενα testbench είναι πάντοτε ιεραρχικό.
  - Τα σήματα που οδηγούν τις εισόδους του υπό έλεγχο κυκλώματος θα πρέπει να δηλωθούν σα τύπου reg.
  - Οι αποκρίσεις του υπό έλεγχο κυκλώματος μπορούν να οδηγούνται σε σήματα τύπου wire.

Initial Statements

– Είναι ένα block εντολών που εκτελείται μόνο μία φορά.
– Ολα τα initial blocks ξεκινούν να εκτελούνται παράλληλα τη χρονική στιγμή 0 της εξομοίωσης.




```
initial
  begin
        A=0; B=0; C=0;
  #100  A=1; B=1; C=1;
  #100  A=0;
  end
```

Χρήσιμες Συναρτήσεις

- $display ("%d %b %b", S, C, I);
- $time – μεταβλητή που κρατάει τη χρονική στιγμή εξομοίωσης
- $write – δεν αλλάζει γραμμή
- $monitor – παρακολουθεί τις αλλαγές μιας μεταβλητής
- $finish – τερματίζει την εξομοίωση
- $stop – σταματά την εξομοίωση
- $random – γεννήτρια τυχαίων αριθμών

Μεγαλύτερα συνδυαστικά κυκλώματα

- Μέχρι τώρα έχουμε χρησιμοποιήσει στα λογικά μας διαγράμματα πύλες και τη διασύνδεσή τους.
- Κάθε είδος πύλης περιέχεται σε ολοκληρωμένα μικρού βαθμού ολοκλήρωσης (SSI).
- Υπάρχουν όμως λογικές συναρτήσεις που τις χρησιμοποιούμε συχνά και με τις οποίες μπορούμε πολύ πιο εύκολα να φτιάξουμε μεγαλύτερα κυκλώματα.
- Στο εμπόριο υπάρχουν επίσης ολοκληρωμένα τα οποία μας δίνουν αυτές τις συναρτήσεις off-the-self.
- Τα ολοκληρωμένα αυτά ενσωματώνουν σημαντικά μεγαλύτερο αριθμό τρανζίστορ.
- Ανήκουν δηλαδή στη κατηγορία των ολοκληρωμένων μεσαίας κλίμακας ολοκλήρωσης (MSI).
- Παρακάτω εξετάζουμε ποια είναι αυτά τα σύνθετα κυκλώματα και πως χρησιμοποιώντας τα μπορούμε να πάρουμε χρήσιμα λογικά κυκλώματα.

MSI που θα μας απασχολήσουν

Αριθμητικά κυκλώματα :

- Ημιαθροιστής
- Πλήρης αθροιστής
- Παράλληλος Αθροιστής / Αφαιρέτης
  - Με διάδοση κρατουμένου
  - Με πρόβλεψη κρατουμένου
- Συγκριτής
- Πολλαπλασιαστής

Κωδικοποίησης / Αποκωδικοποίησης / Πολυπλεξίας / Αποπλεξίας

Κωδικοποιητής

Κωδικοποιητής προτεραιότητας

Αποκωδικοποιητής / Αποπλέκτης

Πολυπλέκτης

Ημι-Αθροιστής (Half Adder)

1.Καθορισμός προβλήματος: κύκλωμα που να προσθέτει δύο δυαδικά ψηφία.

2.Πλήθος εισόδων/εξόδων: 2 είσοδοι – 2 έξοδοι.

3.Ονομασία εισόδων/εξόδων: έστω x, y οι δύο είσοδοι (προσθετέοι) και C (κρατούμενο), S (άθροισμα) οι δύο έξοδοι.

4.Πίνακας αλήθειας:

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Ημι-Αθροιστής

(α)

$S = x'y + xy'$

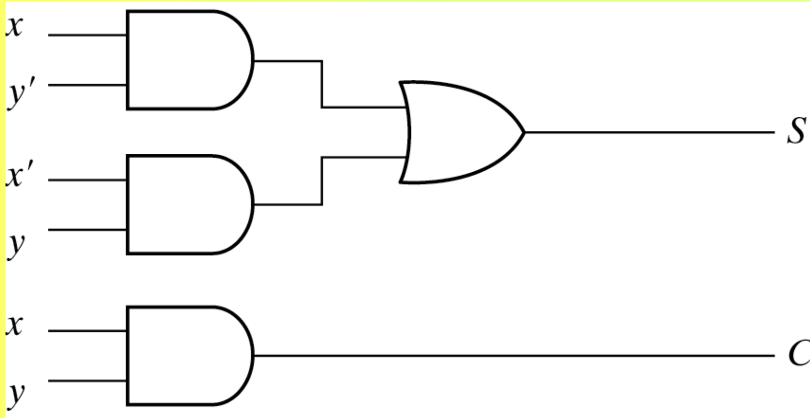
$C = xy$

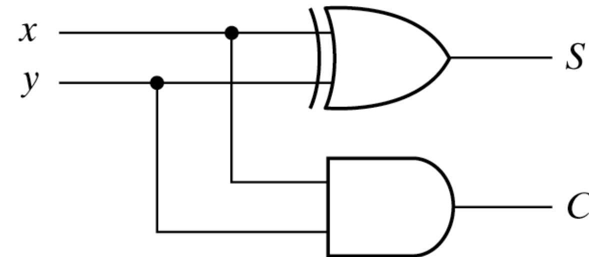
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(β)

$S = x \oplus y$

$C = xy$






```
module HA1 (x, y, S, C) ;
 input x, y;
 output S, C ;

 assign S = x ^ y;
 assign C = x & y;
endmodule
```

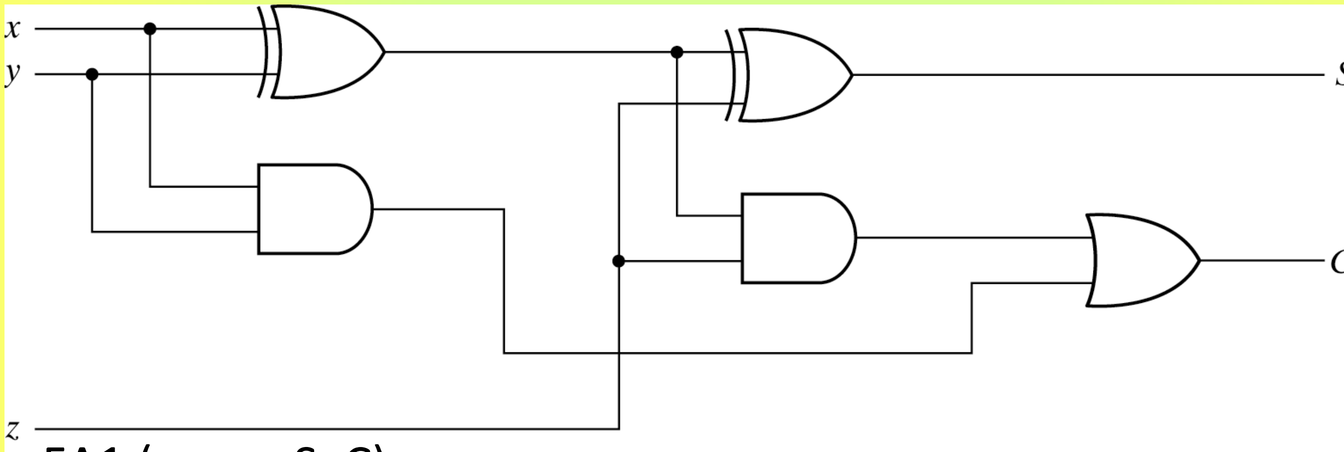
```
module HA1 (x, y, S, C) ;
 input x, y;
 output S, C ;

 assign {C, S} = x + y;
endmodule
```

Πλήρης-Αθροιστής

- Είναι $S = (x \oplus y) \oplus z$
- Επίσης ισχύει

$$C = xy + yz + xz = xy + z\,(x + y) = xy + z\,(x'y + xy' + xy) =$$

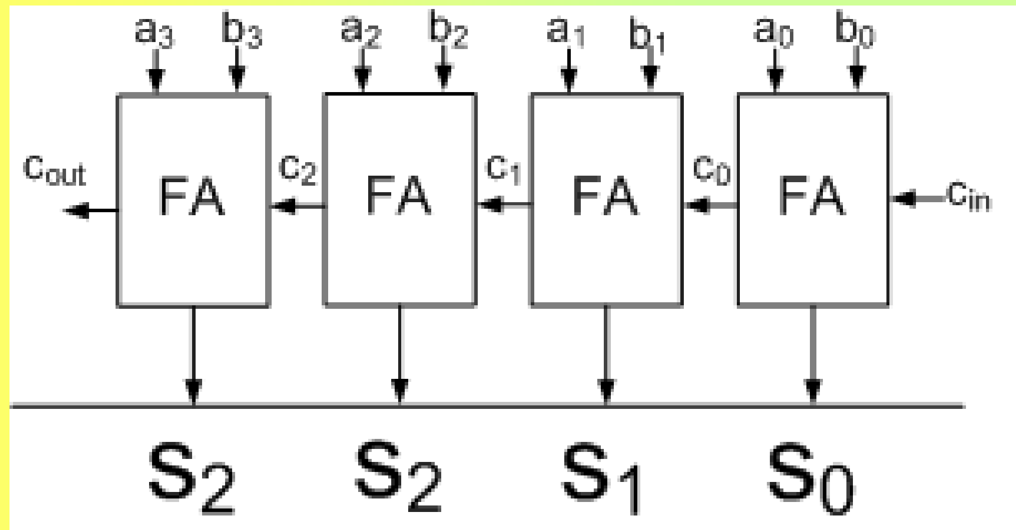
$$xy + zxy + z\,(x'y + xy') = xy + z\,(x \oplus y)$$

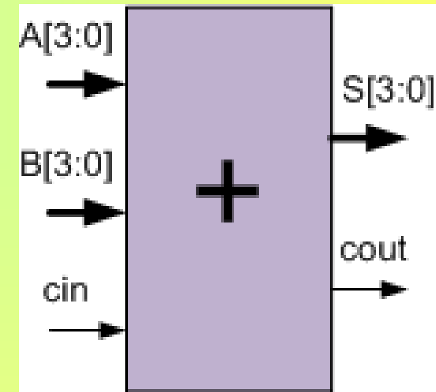



```
module FA1 (x, y, z, S, C) ;
  input x, y, z;
  output S, C ;

  assign S = x ^ y ^ z;
  assign C = (x & y) | (x & z) | (y &z);
endmodule
```

```
module FA2 (x, y, z, S, C) ;
  input x, y;
  output S, C ;

  assign {C, S} = x + y + z;
endmodule
```

Παράλληλος δυαδικός αθροιστής με διάδοση κρατουμένου





Υλοποίηση με συναρτήσεις: Πίνακας αλήθειας με 9 εισόδους και 2^9=512 καταστάσεις.

```
module PA4 (A, B, S, Cin, Cout) ;
  input [3:0] A, B;
  input Cin;
  output [3:0] S ;
  output Cout;

  assign {Cout, S} = A + B + {3'd0,Cin};
endmodule
```

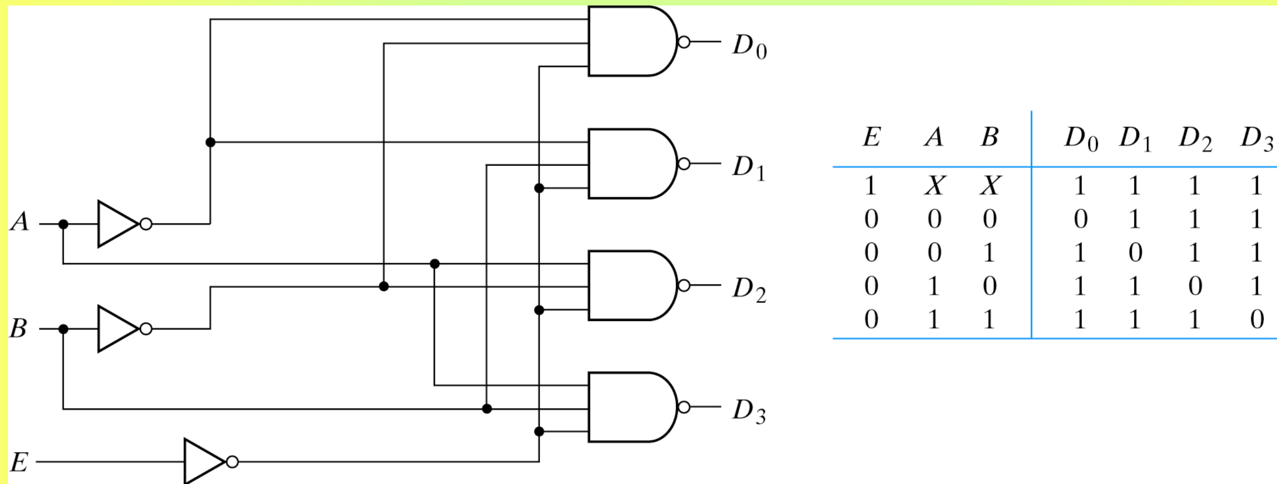
Παράλληλος Αθροιστής/Αφαιρέτης

```
module PAS4 (M, A, B, S, C4) ;
  input [3:0] A, B;
  input M;
  output [3:0] S ;
  output C4;

  assign {C4, S} = M ? A + ~B + {3'd0,1} : A + B;
endmodule
```

Αποκωδικοποιητής με Είσοδο Επίτρεψης

Ο αποκωδικοποιητής μπορεί να παράγει συμπληρωματικές εξόδους.

Ο αποκωδικοποιητής μπορεί να έχει είσοδο επίτρεψης. (Σε αυτή τη περίπτωση συνήθως χρησιμοποιούμε την ορολογία αποπλέκτης).

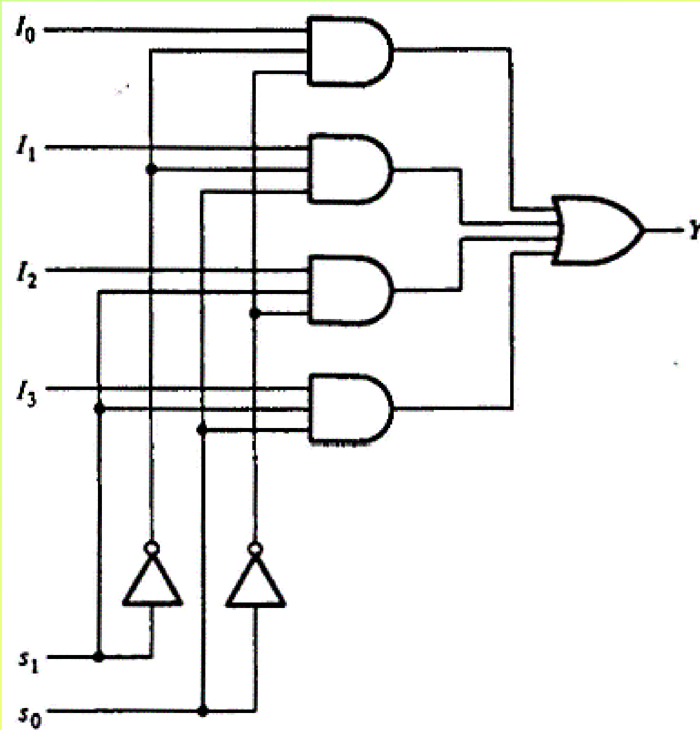


E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

```
module dec4EN_AL (D, E, A, B);
  input      E, A, B;
  output   [3:0] D;

   assign D = ~E ? 4'hF : (~A & ~B)? 4'hE : (~A & B) ? 4'hD : (A & ~B) ? 4'hB : 4'h7;
endmodule
```

Παράδειγμα

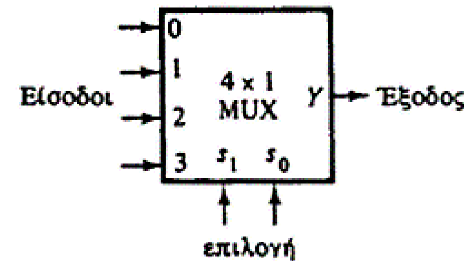
Πολυπλέκτης 4-σε-1



(α) Λογικό διάγραμμα

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(β) Πίνακας της συνάρτησης



(γ) Σχηματικό διάγραμμα

```
module mux4_2_1(I, S, Y);
  input [3:0] I;
  input [1:0] S;
  output Y;
  assign Y = (&S) ? I[3] : ~(|I) ? I[0] : S[1] ? I[2] : I[1];
endmodule
```

Για το σπίτι

Ασκηση 1

Σας δίδονται παράλληλοι αθροιστές των 3 δυαδικών ψηφίων με είσοδο κρατουμένου. Φτιάξτε έναν παράλληλο αθροιστή των 10 δυαδικών ψηφίων.

Ασκηση 2

Προτείνετε ένα ελάχιστο (σε αριθμό απαιτούμενων MSI) κύκλωμα που στην είσοδό του να δέχεται τον αριθμό A που σε δυαδική αναπαράσταση έχει 7 δυαδικά ψηφία, αναπαρίσταται δηλαδή ως $a_6a_5a_4a_3a_2a_1b_0$ και παράγει 2 εξόδους. Η πρώτη μας δείχνει τον αριθμό των 1 της δυαδικής παράστασης του A ενώ η άλλη τον αριθμό των 0.

Ασκηση 3

Προτείνετε ένα ελάχιστο (σε αριθμό απαιτούμενων MSI) κύκλωμα που στην είσοδό του να δέχεται τον αριθμό B που σε δυαδική αναπαράσταση έχει 2 δυαδικά ψηφία, αναπαρίσταται δηλαδή ως b_1b_0 και στην έξοδό του να παράγει το 3B+2.

VHDL

Ροή Σχεδίασης

- Πριν περάσουμε σε περιγραφή της γλώσσας VHDL είναι χρήσιμο να δούμε το περιβάλλον και τη ροή της σχεδίασης. Τα βήματα μιας σχεδίασης βασισμένης σε VHDL μπορούν να ομαδοποιηθούν σε front-end και back-end.

Σχεδίαση

- Αρχικά ψηφιακά κυκλώματα μπορούν να σχεδιασθούν με το κατάλληλο λογισμικό. Μεγαλύτερα κυκλώματα μπορούν να σχεδιασθούν ιεραρχικά από μικρότερα κυκλώματα (δομικές μονάδες). Λεπτομέρειες της σχεδίασης μπορούν να συμπληρωθούν αργότερα χρησιμοποιώντας τη γλώσσα VHDL (text based) .

Συγγραφή

- Στο επόμενο βήμα γράφεται ο κώδικας VHDL που περιγράφει επακριβώς τα δομικά
- στοιχεία, τον τρόπο αλληλεπίδρασης μεταξύ τους και λεπτομέρειες της εσωτερικής
- τους δομής.

Compilation

- Από τη στιγμή που έχει ολοκληρωθεί η συγγραφή του κώδικα, αυτός πρέπει να
- περάσει από τη φάση της συμβολομετάφρασης (compilation). Ένας VHDL compiler αναλύει τον κώδικα για συντακτικά λάθη και ελέγχει τη συμβατότητα του με τμήματα του κώδικα από τον οποίο εξαρτάται. Επίσης, στη φάση αυτή δημιουργούνται και οι πληροφορίες που είναι απαραίτητες για την φάση της προσομοίωσης του κυκλώματος.

Προσομοίωση

- Στη φάση αυτή, με τη βοήθεια της εφαρμογής VHDL Simulator, ορίζονται διάφορες
- τιμές για τις εισόδους του κυκλώματος και προσδιορίζονται οι τιμές των εξόδων, χωρίς να κατασκευαστεί το φυσικό κύκλωμα. Για μικρά κυκλώματα είναι αρκετό να
- δοθούν διάφορες τιμές εισόδου και να παρατηρηθεί η έξοδος χειροκίνητα.
- Σε μεγάλα όμως κυκλώματα, η VHDL παρέχει τη δυνατότητα δημιουργίας ενός test-bench, που θέτει αυτόματα διάφορες τιμές στις εισόδους και τις συγκρίνει με την αναμενόμενη τιμή εξόδου.

Verification

- Η διαδικασία ης προσομοίωσης είναι μέρος μιας ευρύτερης διαδικασίας που λέγεται verification και έχει σαν σκοπό να ελέγξει διεξοδικά την ορθή λειτουργία του κυκλώματος. Μεγάλο μέρος της προσπάθειας καταβάλλεται στο καθορισμό σεναρίων που ελέγχουν το κύκλωμα σε μεγάλο εύρος λογικών συνθηκών.

Functional Verification.

- Η διαδικασία αυτή μπορεί να διακριθεί σε δύο μέρη. Την functional verification όπου η λειτουργία του κυκλώματος ελέγχεται ανεξάρτητα από το χρόνο. Τόσο οι καθυστερήσεις στις πύλες όσο και όλες οι υπόλοιπες παράμετροι χρόνου θεωρούνται μηδέν.

Timing Verification

- Στο δεύτερο μέρος, το timing verification, ελέγχεται η λειτουργία του κυκλώματος λαμβάνοντας υπόψη την αναμενόμενη χρονική καθυστέρηση των πυλών καθώς και τς χρονικές απαιτήσεις σειριακών κυκλωμάτων όπως τα flip-flops. Είναι σύνηθες να ολοκληρώνεται η διαδικασία functional simulation, προτού αρχίσουν τα βήματα που αναφέρονται σαν back-end. Όμως η ικανότητα να πραγματοποιήσουμε με ακρίβεια timing simulation στο σημείο αυτό είναι περιορισμένη, μια και το αποτέλεσμα είναι ισχυρά εξαρτώμενο από τις επόμενες διαδικασίες της Σύνθεσης (synthesis) και της
- Προσαρμογής (Fitting).

Σύνθεση (Synthesis)

- Κατά τη διάρκεια της σύνθεσης η περιγραφή του κυκλώματος στη VHDL μεταφράζεται δομικών στοιχείων που μπορούν να συναρμολογηθούν στην τεχνολογία
- για την οποία προορίζεται το κύκλωμα. Για παράδειγμα, για ένα FPGA δημιουργείται ένα σύνολο πυλών και ένα σύνολο από συνδέσεις (netlist) που περιγράφει πώς οι πύλες θα συνδεθούν μεταξύ τους.

Προσαρμογή (Fitting, Place & Route)

- Στη διαδικασία αυτή, το κατάλληλο λογισμικό (fitter) προσαρμόζει τα δομικά στοιχεία που δημιουργήθηκαν στο προηγούμενο βήμα στους διαθέσιμους πόρους της συσκευής για την οποία προορίζεται το κύκλωμα. Ο σχεδιαστής στο βήμα αυτό μπορεί να εισάγει περιορισμούς τόσο για την τοποθέτηση των δομικών στοιχείων στη συσκευή όσο και για την εκχώρηση των pins των εισόδων και εξόδων του κυκλώματος. Το τελευταίο βήμα είναι η timing verification του προσαρμοσμένου κυκλώματος, όπου μπορούν να ληφθούν υπόψη οι χρονικοί περιορισμοί που εισάγουν οι πύλες, το μήκος των καλωδίων, ο ηλεκτρικό φόρτος του κυκλώματος κτλ. Σε αυτό το βήμα ελέγχονται τα ίδια σενάρια με την περίπτωση του functional verification, γνωρίζοντας όμως με ακρίβεια το τρόπο με τον οποίο θα δομηθεί το κύκλωμα στο PGA.

Δομή προγράμματος VHDL

- Η VHDL είναι μια γλώσσα που δημιουργήθηκε έχοντας αρχές του δομημένου προγραμματισμού. Η κύρια ιδέα είναι να ορίζει τις εισόδους και εξόδους ενός κυκλώματος, κρύβοντας την εσωτερική του δομή. Στη VHDL με τη δήλωση entity ορίζονται οι είσοδοι και έξοδοι μιας αυτοτελούς μονάδας (module), ενώ με τη δήλωση architecture περιγράφεται η εσωτερική δομή και λειτουργία της αυτοτελούς μονάδας.
- Στο αρχείο ενός προγράμματος VHDL η δήλωση entity και architecture είναι διαχωρισμένες. Ακολουθεί ένα παράδειγμα ενός προγράμματος σε VHDL

```
entity Inhibit is -- also known as 'BUT-NOT'
port (X,Y: in BIT; --as in 'X but not Y' Z: out BIT); --(see [Klir, 1972])
end Inhibit;
architecture Inhibit_arch of Inhibit is
begin
Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

- Όπως σε κάθε γλώσσα, έτσι και στη VHDL υπάρχει μια σειρά από δεσμευμένες λέξεις. Στο ανωτέρω πρόγραμμα τέτοιες είναι οι entity, port, is, in , out, end, architecture, begin, when, else και not.
- Με τη δήλωση entity ορίζεται το όνομα της αυτοτελούς μονάδος καθώς και οι είσοδοι και έξοδοι της (Με τη δεσμευμένη λέξη port) . Τα X, Y, Z αποτελούν τα σήματα εισόδου / εξόδου στη μονάδα. Η κατεύθυνση τους προσδιορίζεται με τη χρήση των λεκτικών in, out και inout.
- Ορίζεται επίσης και το είδος του σήματος π.χ. bit, real, integer, boolen κτλ.
- Με τη δήλωση architecture ορίζεται η εσωτερική δομή της μονάδας.
- Τα σήματα (εισόδου & εξόδου) πηγάζουν από τη δήλωση entity που έχει προηγηθεί. Η architecture μπορεί να περιλαμβάνει, επίσης, σήματα και δηλώσεις που είναι τοπικά (local) κατ’ αναλογία με τις γνωστές γλώσσες υψηλού επιπέδου.

Σε αντιστοιχία με τις γλώσσες υψηλού επιπέδου, μια VHDL function, δέχεται ένα σύνολο από ορίσματα και επιστρέφει ένα αποτέλεσμα.

Η δομή μιας συνάρτησης (function) VHDL είναι ως εξής:

```
function function-name (signal-names : signal-type;
    signal-names : signal-type;
    ...
    signal-names : signal-type
   ) return return-type is
   type declarations
   constant declarations
   function definitions
   procedure definitions
    begin
    sequential-statement
     ...
    sequential-statement
    end function-name;
```

- Μια VHDL procedure (ρουτίνα) είναι παρόμοια με τη συνάρτηση (function) με τη διαφορά ότι δεν επιστρέφει κάποιο αποτέλεσμα. Μια ρουτίνα (procedure) δέχεται ορίσματα τύπου out ή inout. Οπότε με το τρόπο αυτό είναι εφικτό να επιστρέψει ένα αποτέλεσμα.

Libraries and Packages

- Μια βιβλιοθήκη (library) VHDL αποτελεί το χώρο όπου ο VHDL compiler αποθηκεύει πληροφορίες για μια σχεδίαση ενός project, συμπεριλαμβάνοντας και ενδιάμεσα αρχεία που χρησιμοποιούνται κατά την ανάλυση, προσομοίωση και σύνθεση του κυκλώματος. Για την τρέχουσα σχεδίαση σε VHDL ο compiler δημιουργεί αυτόματα και χρησιμοποιεί στη συνέχεια τη βιβλιοθήκη που ονομάζεται “work”. Η βιβλιοθήκη “work” (συνήθως ένας υποκατάλογος στο κατάλογο του project) εν περιλαμβάνει όλες τις πληροφορίες που είναι απαραίτητες για ένα συγκεκριμένο project. Ακόμη και πολύ μικρά σχέδια κυκλωμάτων είναι δυνατό να χρησιμοποιούν ορισμούς από τη βιβλιοθήκη της IEEE. Για να συμπεριληφθεί η βιβλιοθήκη αυτή στον κώδικα VHDL απαιτείτε η δήλωση: library ieee; Η δήλωση “library work;” υπονοείτε ότι περιλαμβάνεται στην αρχή κάθε αρχείου VHDL.
- Σε μια βιβλιοθήκη περιλαμβάνονται αυτοτελείς μονάδες (entities & architectures), όχι όμως και ορισμοί μεταβλητών, τύπων σημάτων κτλ. Τέτοιες πληροφορίες αποθηκεύονται σε VHDL packages (πακέτα). Πρόσβαση στους ορισμούς ενός package γίνεται με τη χρήση του λεκτικού “use”. Για παράδειγμα, για να χρησιμοποιηθούν όλοι οι ορισμοί του πακέτου IEEE standard 1164 απαιτείται η δήλωση:

```
use ieee.std_logic_1164.all;
```

Processes

- Η δήλωση process στη VHDL είναι ο κυριότερος τρόπος περιγραφής σειριακών λειτουργιών (π.χ. μνήμης, registeres circuits). Σε τέτοιες περιπτώσεις η κυριότερη δομή της process είναι:

```
architecture arch_name of end_name is
begin
 process_name: process(sensitivity_list)
local declaration;
local declaration;
….
begin
sequential statement;
sequential statement;
……
end process;
end arch_name;
```

- Η sensitivity_list είναι προαιρετική και δηλώνει τα σήματα στα οποία όταν ανιχνευθεί μια αλλαγή θα εκτελεσθεί η process. Καθίσταται απαραίτητη στις περιπτώσεις όπου δεν υπάρχουν εντολές wait στην process για να αναστείλουν την εκτέλεση της σε ορισμένα σημεία. Ένα flip-flop είναι ένα πολύ καλό παράδειγμα που περιγράφεται με μια process. Παραμένει αδρανές χωρίς να αλλάζει κατάσταση να συμβεί κάποιο σημαντικό γεγονός (π.χ. αλλαγή του ρολογιού ή κάποιο γεγονός reset) το οποίο το θέτει σε λειτουργία και εν γένει το οδηγεί σε άλλη κατάσταση.

- Στο παράδειγμα που ακολουθεί, παρουσιάζεται η λειτουργία της process για το κύκλωμα barrel sifter reg:

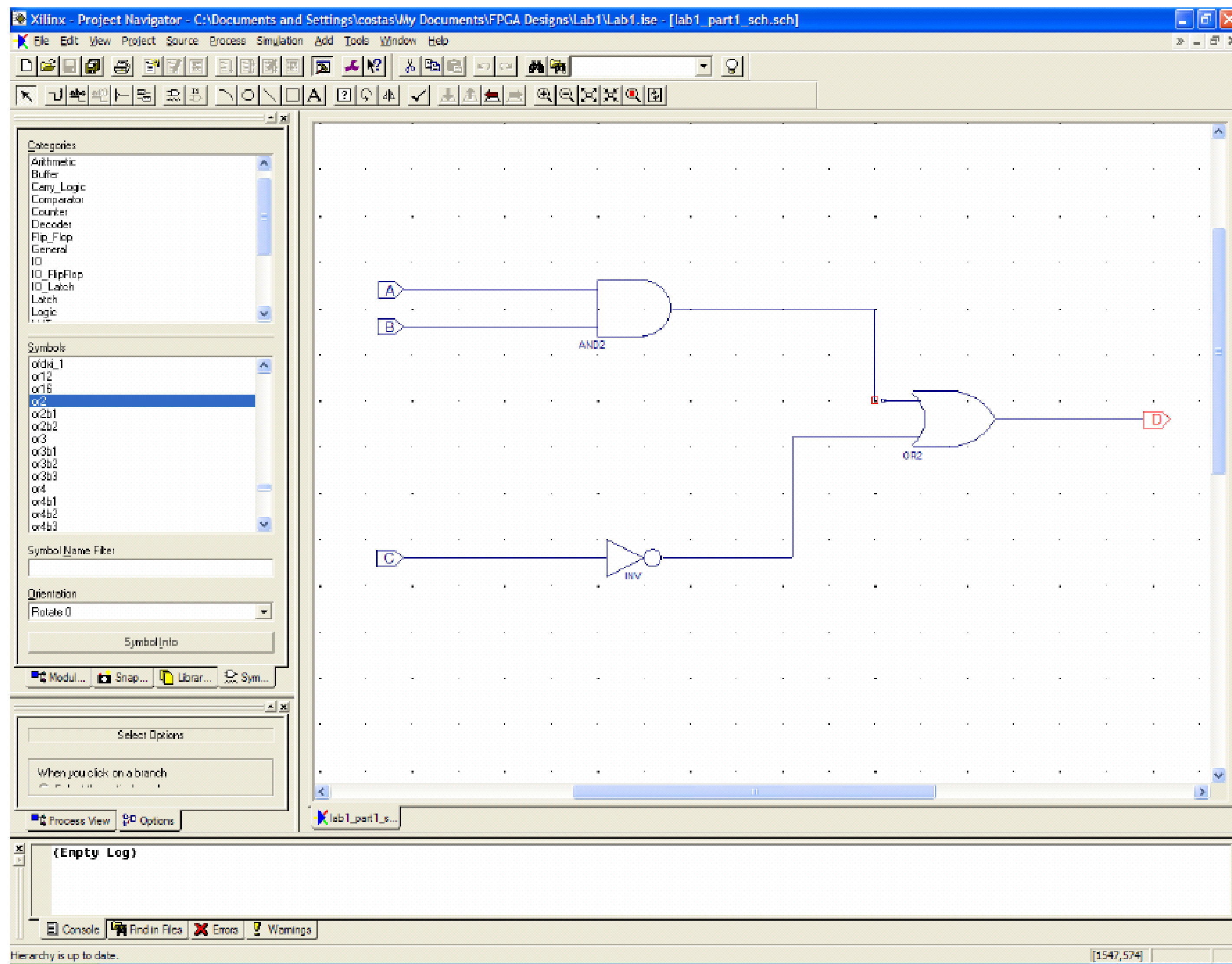
```
process(Rst, Clk)
variable Qreg: std_logic_vector(0 to 7);
begin
if Rst='1' then --async reset
Qreg:="00000000";
elseif (Clk = '1' and Clk'event) then
if (Load = '1') then
Qreg := Data;
else
Qreg := Qreg(1 to 7 ) & Qreg(0);
end if;
end if;
Q <=Qreg;
end process;
```

- Τα σήματα Rst και Clk είναι τα μόνα που μπορούν να ενεργοποιήσουν την process. Αν δεν υπάρξει κανένα γεγονός σε αυτά, τότε η process βρίσκεται σε κατάσταση αναμονής. Αν υπάρχει κάποιο γεγονός στην είσοδο Rst που τη φέρνει στη κατάσταση ΄1΄η πρώτη δήλωση if θα εκτελεστεί καιη μεταβλητή Qreg θα λάβει τη τιμή "00000000". Αν συμβεί κάποιο γεγονός στο σήμα Clk, τότε η διεργασία θα εκτελεσθεί ξανά. Αν το Rst εξακολουθεί να έχει τη τιμή '1', θα εκτελεσθεί το πρώτο if όπως και νωρίτερα. Αν όμως το Rst δεν έχει τη τιμή '1' η έκφραση (Clk = '1' and Clk'event) θα προσδιορισθεί. Για να ανιχνευθεί η μεταβολή από τη θέση '0' στη θέση '1' αρκεί η δήλωση Clk = '1'. Όμως αν η process εκτελέστηκε εξαιτίας κάποιου γεγονότος στο Rst, π.χ. τη πτώση από '1' σε '0' δεν επιθυμούμε να εκτελεσθεί το μέρος του κώδικα που αφορά γεγονός στο Clk. Για να το διασφαλίσουμε έχουμε προσθέσει τη δήλωση Clk'event.

Σήματα και Μεταβλητές

- Δύο είναι οι κύριοι τύποι των αντικειμένων που συναντάμε στην VHDL για τη μεταφορά δεδομένων: σήματα και μεταβλητές. Σε γενικές γραμμές οι μεταβλητές χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ σειριακών διεργασιών (processes, procedures και functions), ενώ τα σήματα χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ των παράλληλων στοιχείων του σχεδίου του κυκλώματος (π.χ. δύο processes) Είναι χρήσιμο να σκεφτόμαστε τα σήματα σαν καλώδια (όπως στο σχέδιο του κυκλώματος) και τις μεταβλητές σαν προσωρινές θέσεις μνήμης.

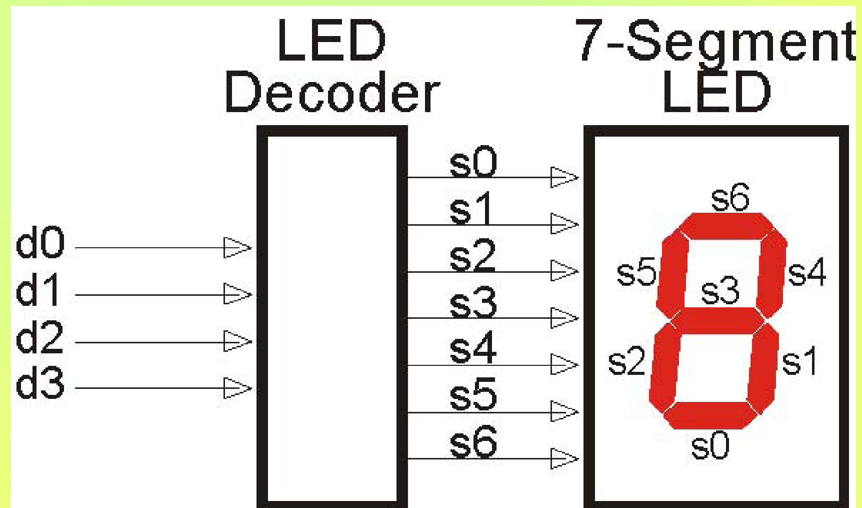
Περιβάλλον υλοποίησης



Κύκλωμα αποκωδικοποιητή led (led decoder) .

- Το κύκλωμα δέχεται 4 – bit σαν είσοδο και έχει 7- bit σαν έξοδο με τα οποία οδηγούμε το led

Four-bit Input	Hex Digit	LED Display
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	A
1011	B	b
1100	C	c
1101	D	d
1110	E	E
1111	F	F

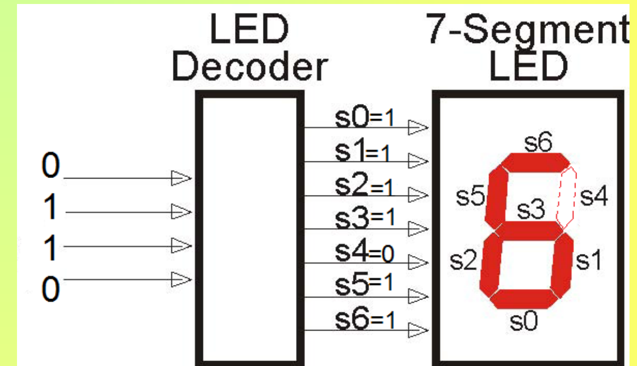


Είσοδος	Έξοδος	LED Display
0000	1110111	0
0001	0010010	1
0010	1011101	2
0011	1011011	3
0100	0111010	4
0101	1101011	5
0110	1101111	6
0111	1010010	7
1000	1111111	8
1001	1111011	9
1010	1111110	A
1011	0101111	B
1100	1100101	C
1101	0011111	D
1110	1101101	E
1111	1101100	F

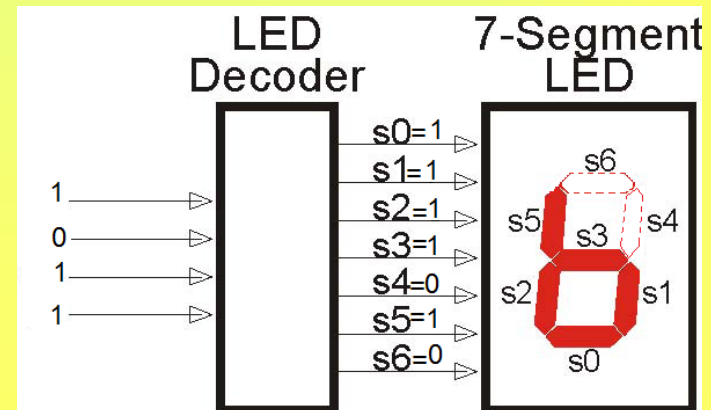
Κώδικας σε VHDL

- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL;
- use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity leddcd is
- Port (d : in STD_LOGIC_VECTOR (3 downto 0);
- s : out STD_LOGIC_VECTOR (6 downto 0));
- end leddcd;
- architecture Behavioral of leddcd is
- begin
- s<="1110111" when d="0000" else
- "0010010" when d="0001" else
- "1011101" when d="0010" else
- "1011011" when d="0011" else
- "0111010" when d="0100" else
- "1101011" when d="0101" else
- "1101111" when d="0110" else
- "1010010" when d="0111" else
- "1111111" when d="1000" else
- "1111011" when d="1001" else
- "1111110" when d="1010" else
- "0101111" when d="1011" else
- "1100101" when d="1100" else
- "0011111" when d="1101" else
- "1101101" when d="1110" else
- "1101100";
- end Behavioral;

Παράδειγμα υλοποίησης του αριθμού 6



Παράδειγμα υλοποίησης του $B_{16} = 11_{10}$

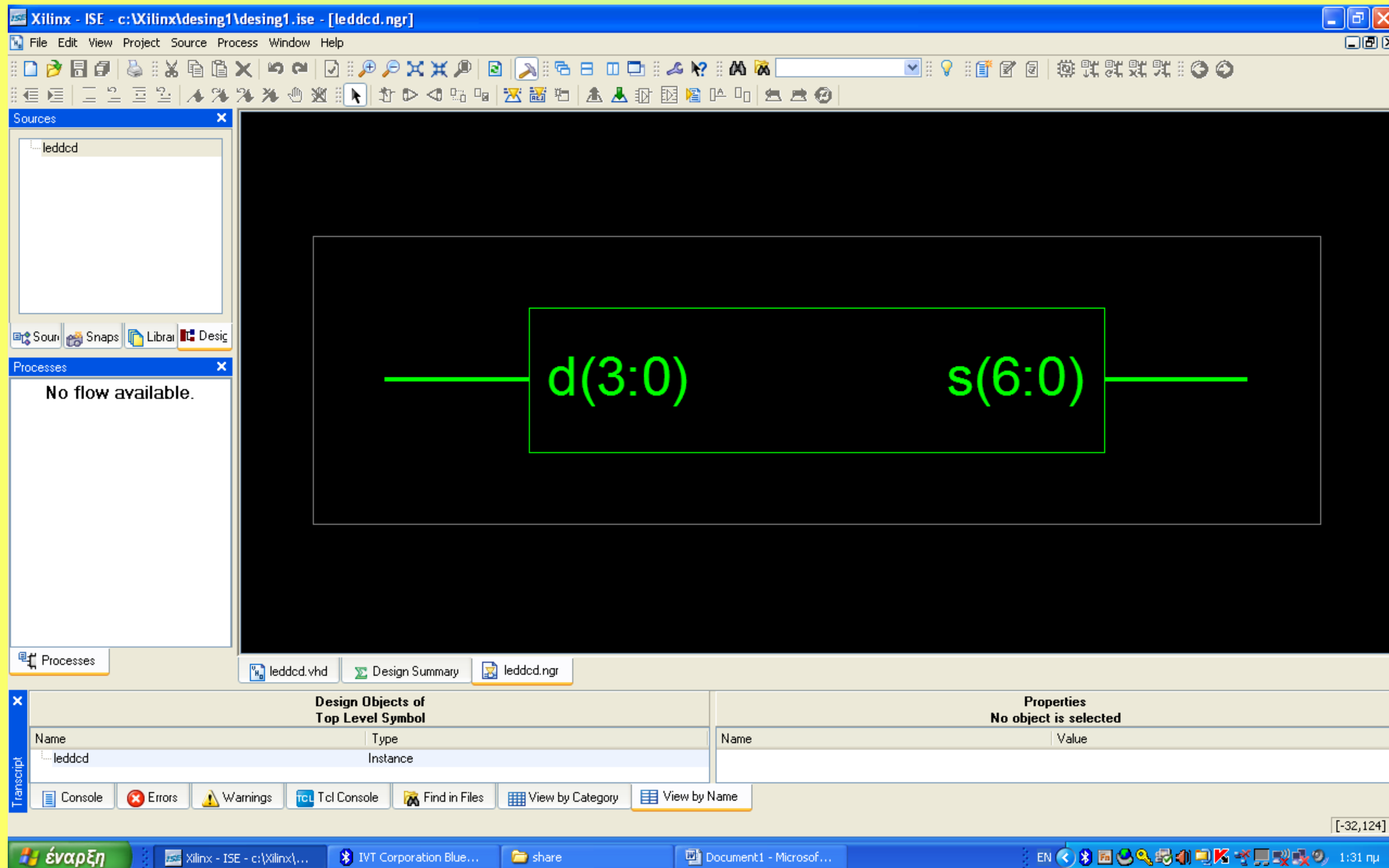


Περιβάλλον υλοποίησης

Xilinx - ISE - c:\Xilinx\desing1\desing1.ise - [leddcd.vhd]

```
--library UNISIM;
--use UNISIM.VComponents.all;

entity leddcd is
    Port ( d : in  STD_LOGIC_VECTOR (3 downto 0);
           s : out  STD_LOGIC_VECTOR (6 downto 0));
end leddcd;

architecture Behavioral of leddcd is

begin
s<= "1110111" when d="0000" else
    "0010010" when d="0001" else
    "1011101" when d="0010" else
    "1011011" when d="0011" else
    "0111010" when d="0100" else
    "1101011" when d="0101" else
    "1101111" when d="0110" else
    "1010010" when d="0111" else
    "1111111" when d="1000" else
    "1111011" when d="1001" else
    "1111110" when d="1010" else
    "0101111" when d="1011" else
    "0001101" when d="1100" else
    "0011111" when d="1101" else
    "1101101" when d="1110" else
    "1101100";


end Behavioral;
```

Σύνθεση


Xilinx - ISE - c:\Xilinx\desing1\desing1.ise - [leddcd.ngr]
File Edit View Project Source Process Window Help
Sources
leddcd
Processes
No flow available.
d(3:0)
s(6:0)
leddcd.vhd
Design Summary
leddcd.ngr
Design Objects of Top Level Symbol
Name
Type
leddcd
Instance
Properties No object is selected
Name
Value
Console
Errors
Warnings
Tcl Console
Find in Files
View by Category
View by Name
[-32,124]
έναρξη

Δημιουργία πλήρη αθροιστή 3-bit σε VHDL

Xilinx - ISE - C:\Xilinx\askisi05\askisi05.ise - [athroistis_3_bit.sch*]

File Edit View Project Source Process Add Tools Window Help

Sources

Sources: Synthesis/Implementat Numbe LU

Hierarchy | Resources | Preserve

- askisi05
- xc2s5...

Source | Snapshc | Librarie | Symbol

Processes

Processes:

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
- Generate Programming File

Processes | Options

XOR2 XOR2 AND2 AND2 OR2

athroistis_3_bit.sch* | Design Summary

```
Started : "Generate Programming File".

Process "Generate Programming File" completed successfully
```

Transcript

Console | Errors | Warnings | Tcl Console | Find in Files

[1184,1320]

έναρξη | Xilinx - ISE - C:\Xilinx\... | Επιφάνεια εργασίας | Τα έγγρ | EN | 2:42 μμ

Δημιουργία πλήρη αθροιστή 3-bit σε VHDL

Xilinx - ISE - C:\Desktop\pliris_athristis\pliris_athristis.ise - [pliris_athristis.vhd]

File Edit View Project Source Process Window Help

Sources

Sources for: Synthesis/Implem Number of: LUTs

Hierarchy

- pliris_athristis
- xc2s50-5tq144
  - pliris_athristis_vhdl - Behavioral (pliris_athristis.vhd)

Sources | Snapshots | Libraries

Processes

- Synthesize - XST
  - View Synthesis Report
  - View RTL Schematic
  - View Technology Schematic
  - Check Syntax
  - Generate Post-Synthesis Simulation Model
    - Post-Synthesis Simulation Model Report
- Implement Design
  - Translate
  - Map
  - Place & Route
- Generate Programming File
  - Programming File Generation Report
  - Generate PROM, ACE, or JTAG File
  - Configure Device (iMPACT)

```
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity pliris_athristis_vhdl is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           g : in  STD_LOGIC;
           s : out  STD_LOGIC;
           c : out  STD_LOGIC);
end pliris_athristis_vhdl;

architecture Behavioral of pliris_athristis_vhdl is

begin
s<=a xor (b xor g);
c<=( a and (b xor g))or (b and g);

end Behavioral;
```

Design Summary | pliris_athristis.vhd

```
Started : "Generate Programming File".

Process "Generate Programming File" completed successfully
```

Console | Errors | Warnings | Tcl Console | Find in Files

Ready — Ln 1 Col 1 CAPS NUM SCRL VHDL

start | pliris_athristis | Xilinx - ISE - C:\Deskt... | Εργαστηριακή άσκησ... | ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ Ψ... | EN | 1:48 πμ

Δημιουργία πολυπλέκτη 4 σε 1.

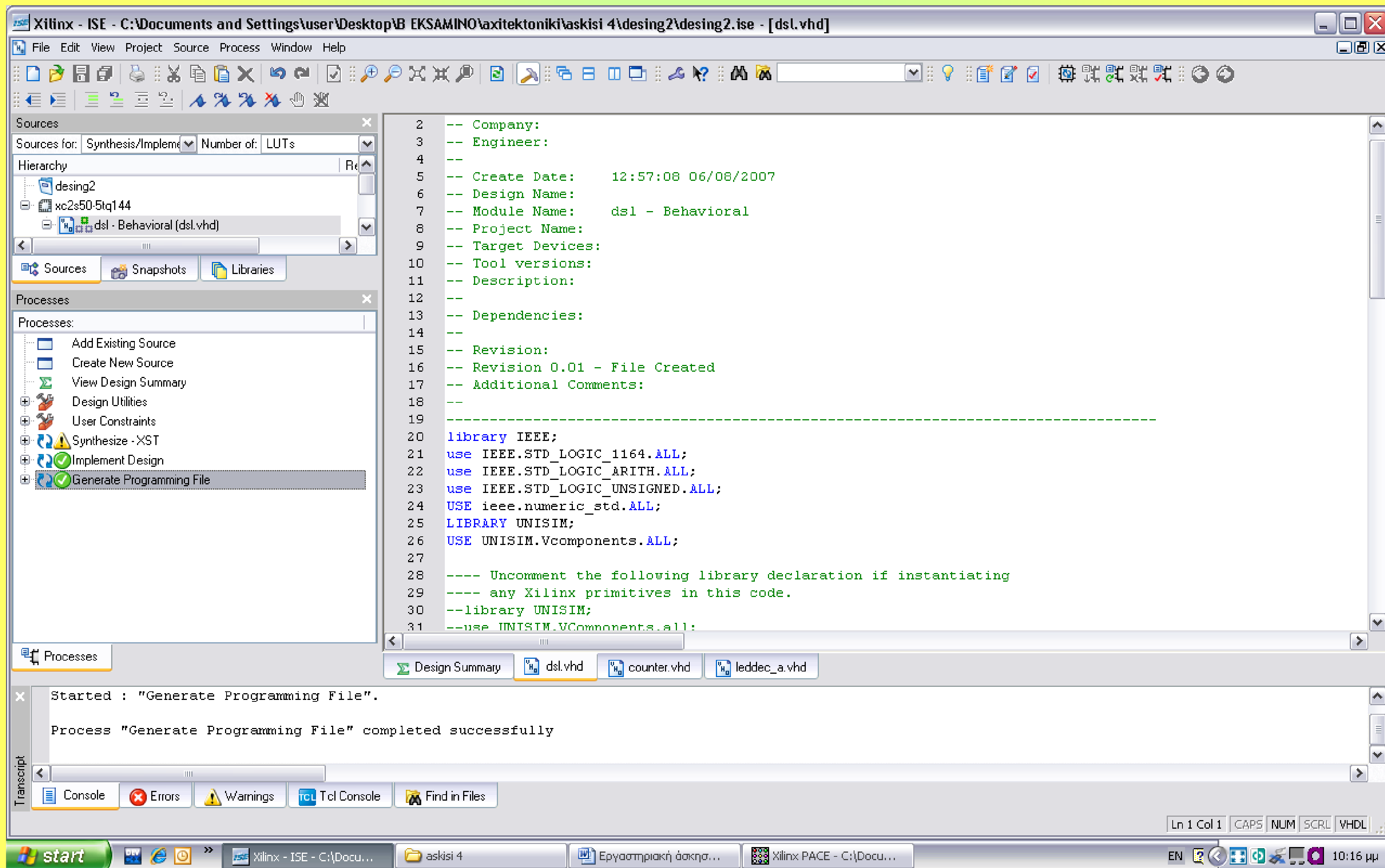
Να σχεδιάσετε έναν πολυπλέκτη 4 σε 1 (γράφοντας ένα απλό πρόγραμμα σε VHDL ή σχεδιάζοντας το σε επίπεδο πυλών). Το κύκλωμα πρέπει να έχει τέσσερις εισόδους (a,b,c,d), δύο γραμμές επιλογών (s0, s1) και μια έξοδο (out) εύρους 1-bit.

Ο πολυπλέκτης είναι ένα συνδυαστικό κύκλωμα, το οποίο επιλέγει τη δυαδική πληροφορία μιας από πολλές γραμμές εισόδου και την κατευθύνει σε μια μοναδική γραμμή εξόδου. Καθεμιά από τις τέσσερις εισόδους, a, b, c και d συνδέεται με μια από τις εισόδους μιας πύλης AND. Οι γραμμές επιλογής s0 και s1 αποκωδικοποιούνται έτσι ώστε να επιλέξουν μόνο μια συγκεκριμένη πύλη AND. Οι έξοδοι των πυλών AND τροφοδοτούν τις εισόδους μιας μοναδικής πύλης OR, η έξοδος της οποίας είναι η έξοδος του κυκλώματος.

Xilinx - ISE - C:\Desktop\poliplekti\poliplekti.ise - [poliplekti.vhd]

```
--
------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity poliplekti is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
           s0 : in  STD_LOGIC;
           s1 : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end poliplekti;

architecture Behavioral of poliplekti is

begin
Y<=(a and (not s1) and (not s0)) or (b and (not s1)and s0) or (c and s1 and (not s0)) or (d and s1 and s0);

end Behavioral;
```

Ιεραρχική σχεδίαση.



Ιεραρχική σχεδίαση.

Xilinx - ISE - C:\Documents and Settings\user\Desktop\B EKSAMINO\axitektoniki\askisi 4\desing2\desing2.ise - [dsl.vhd*]

File Edit View Project Source Process Window Help

Sources

Sources for: Synthesis/Impleme Number of: LUTs

Hierarchy

- Items_1 - counter - Behavioral (counter.vhd)
- Items_2 - leddec_a - Behavioral (leddec_a.vhd)
- dsl.ucf (dsl.ucf)

Sources | Snapshots | Libraries

Processes

Processes:

- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
  - Create Timing Constraints
  - Assign Package Pins
  - Create Area Constraints
  - Edit Constraints (Text)
- Synthesize - XST
- Implement Design
- Generate Programming File

Processes

```
entity dsl is
    Port ( clk : in  STD LOGIC;
           s : out  STD LOGIC VECTOR (6 downto 0));
end dsl;

architecture Behavioral of dsl is
 SIGNAL cnt to dec :  STD LOGIC VECTOR (3 DOWNTO 0);
   SIGNAL in buf   :  STD LOGIC;
   SIGNAL a    :      STD LOGIC VECTOR (6 DOWNTO 0);
   ATTRIBUTE BOX TYPE : STRING;
   COMPONENT counter
      PORT ( clk            :          IN        STD LOGIC;
             count    :            OUT     STD LOGIC VECTOR (3 DOWNTO 0));
   END COMPONENT;
   COMPONENT IBUFG
      PORT ( I    :            IN        STD LOGIC;
             O         :            OUT     STD LOGIC);
   END COMPONENT;
   ATTRIBUTE BOX TYPE OF IBUFG : COMPONENT IS "BLACK BOX";
   COMPONENT leddec a
      PORT ( d   :            IN        STD LOGIC VECTOR (3 DOWNTO 0);
             s         :            OUT     STD LOGIC VECTOR (6 DOWNTO 0));
   END COMPONENT;
   COMPONENT OBUF
      PORT ( I    :            IN        STD LOGIC;
            O        :            OUT     STD LOGIC);
   END COMPONENT;
   ATTRIBUTE BOX TYPE OF OBUF : COMPONENT IS "BLACK BOX";
```

Design Summary | dsl.vhd*

```
Started : "Assign Package Pins".
```

Transcript

Console | Errors | Warnings | Tcl Console | Find in Files

Ln 60 Col 58 CAPS NUM SCRL VHDL

start | Xilinx - ISE - C:\Docu... | askisi 4 | Εργαστηριακή άσκηση... | EN | 9:08 μμ

Ιεραρχική σχεδίαση.

Xilinx - ISE - C:\Documents and Settings\user\Desktop\B EKSAMINO\axitektoniki\askisi 4\desing2\desing2.ise - [dsl.vhd*]

Sources: Items_1 - counter - Behavioral (counter.vhd); Items_2 - leddec_a - Behavioral (leddec_a.vhd); dsl.ucf (dsl.ucf)

Processes: Add Existing Source; Create New Source; View Design Summary; Design Utilities; User Constraints; Create Timing Constraints; Assign Package Pins; Create Area Constraints; Edit Constraints (Text); Synthesize - XST; Implement Design; Generate Programming File

```
   ATTRIBUTE BOX_TYPE OF OBUF : COMPONENT IS "BLACK_BOX";
begin
Items_1 : counter
      PORT MAP (clk=>in_buf, count(3)=>cnt_to_dec(3), count(2)=>cnt_to_dec(2),
      count(1)=>cnt_to_dec(1), count(0)=>cnt_to_dec(0));
   Items_18 : IBUFG
      PORT MAP (I=>clk, O=>in_buf);
   Items_2 : leddec_a
      PORT MAP (d(3)=>cnt_to_dec(3), d(2)=>cnt_to_dec(2), d(1)=>cnt_to_dec(1),
      d(0)=>cnt_to_dec(0), s(6)=>a(6), s(5)=>a(5), s(4)=>a(4), s(3)=>a(3),
      s(2)=>a(2), s(1)=>a(1), s(0)=>a(0));
   Items_4 : OBUF
      PORT MAP (I=>a(0), O=>s(0));
   Items_5 : OBUF
      PORT MAP (I=>a(1), O=>s(1));
   Items_6 : OBUF
      PORT MAP (I=>a(2), O=>s(2));
   Items_7 : OBUF
      PORT MAP (I=>a(3), O=>s(3));
   Items_8 : OBUF
      PORT MAP (I=>a(4), O=>s(4));
   Items_10 : OBUF
      PORT MAP (I=>a(6), O=>s(6));
   Items_9 : OBUF
      PORT MAP (I=>a(5), O=>s(5));
end Behavioral;
```

Started : "Assign Package Pins".

Σχεδιασμός D_Flip_Flop σε VHDL.

Xilinx - ISE - C:\Desktop\D_Flip_Flop\D_Flip_Flop.ise - [D_Flip_Flop.vhd]

File Edit View Project Source Process Window Help

Sources

Sources for: Synthesis/Impleme Number of: LUTs

Hierarchy | Resources | Preserve

D_Flip_Flop
xc2s50-5t...
D...

Sources | Snapshots | Libraries

Processes

Processes:

Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Create Timing Constraints
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
Implement Design
Generate Programming File

Processes

```vhdl
-- Additional Comments:
--
-------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity D_Flip_Flop is
    Port ( rst : in  STD_LOGIC;
           D : in  STD_LOGIC;
           clk  : in  STD_LOGIC;
           Y : out  STD_LOGIC);
end D_Flip_Flop;

architecture Behavioral of D_Flip_Flop is

begin
process(rst,clk)
begin
if (rst='1') then Y<='1';
else if clk='1' then Y<=D;
end if;
end if;
end process;
```

D_Flip_Flop.vhd | Design Summary

```
Started : "Generate Programming File".

Process "Generate Programming File" completed successfully
```

Transcript

Console | Errors | Warnings | Tcl Console | Find in Files

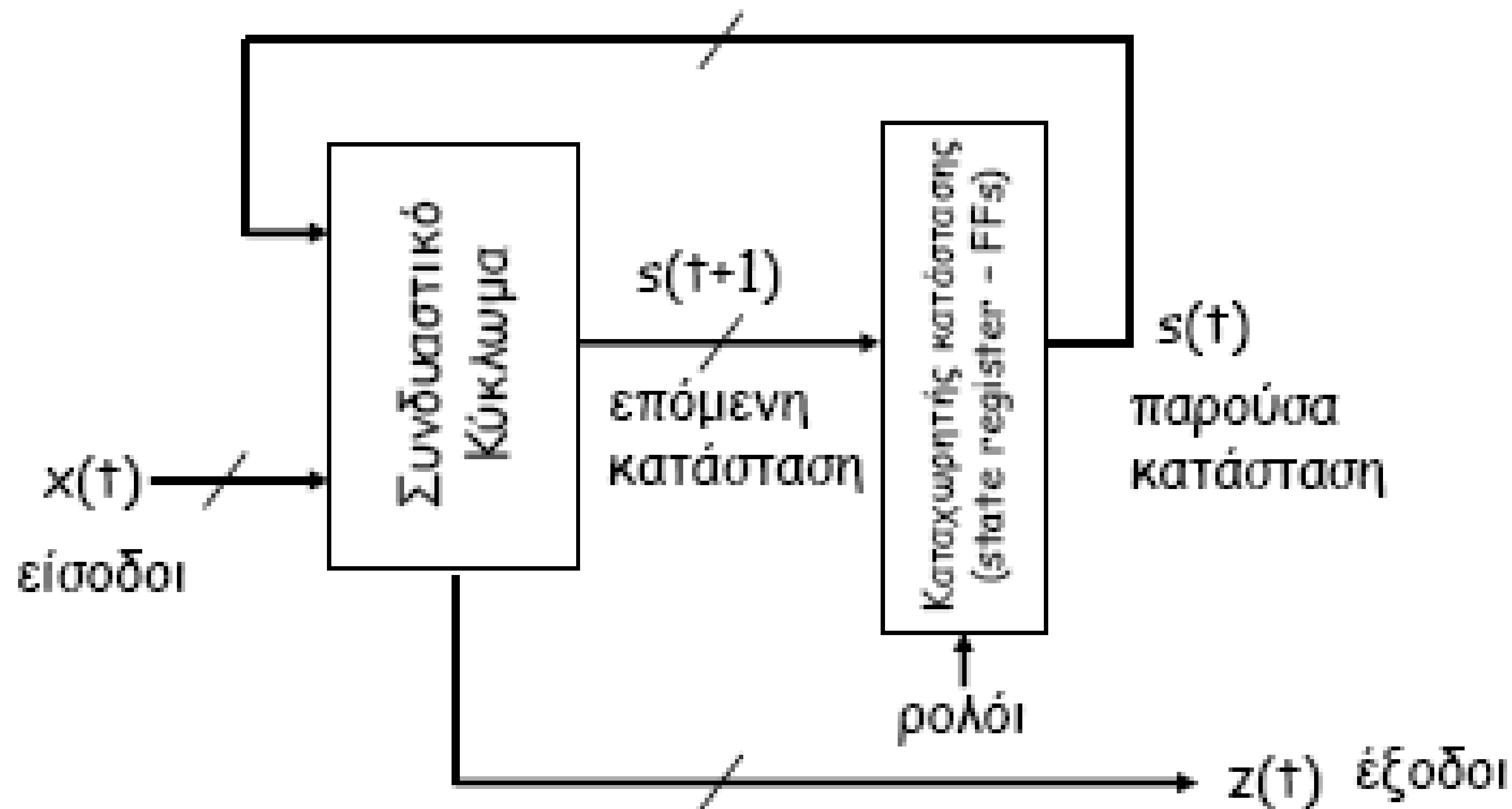
Ln 46 Col 13 | CAPS | NUM | SCRL | VHDL

start | Ergastiriaki askisi 7 - ... | ergasthriaki askisi 6 a... | Xilinx - ISE - C:\Deskt... | axitektoniki | EN | 9:37 μμ

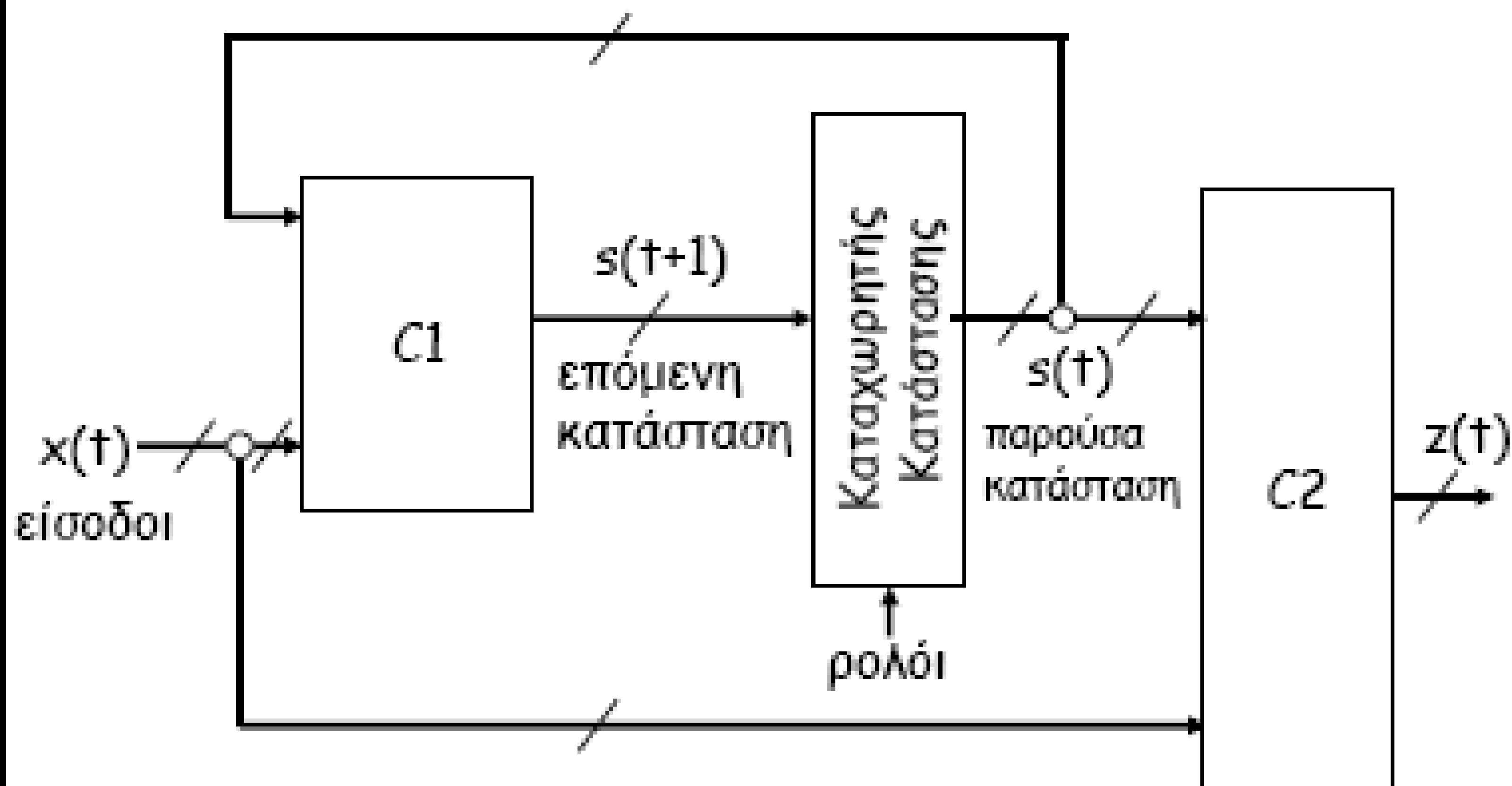
Μηχανές πεπερασμένων καταστάσεων
(Finite State machines-FSMs)

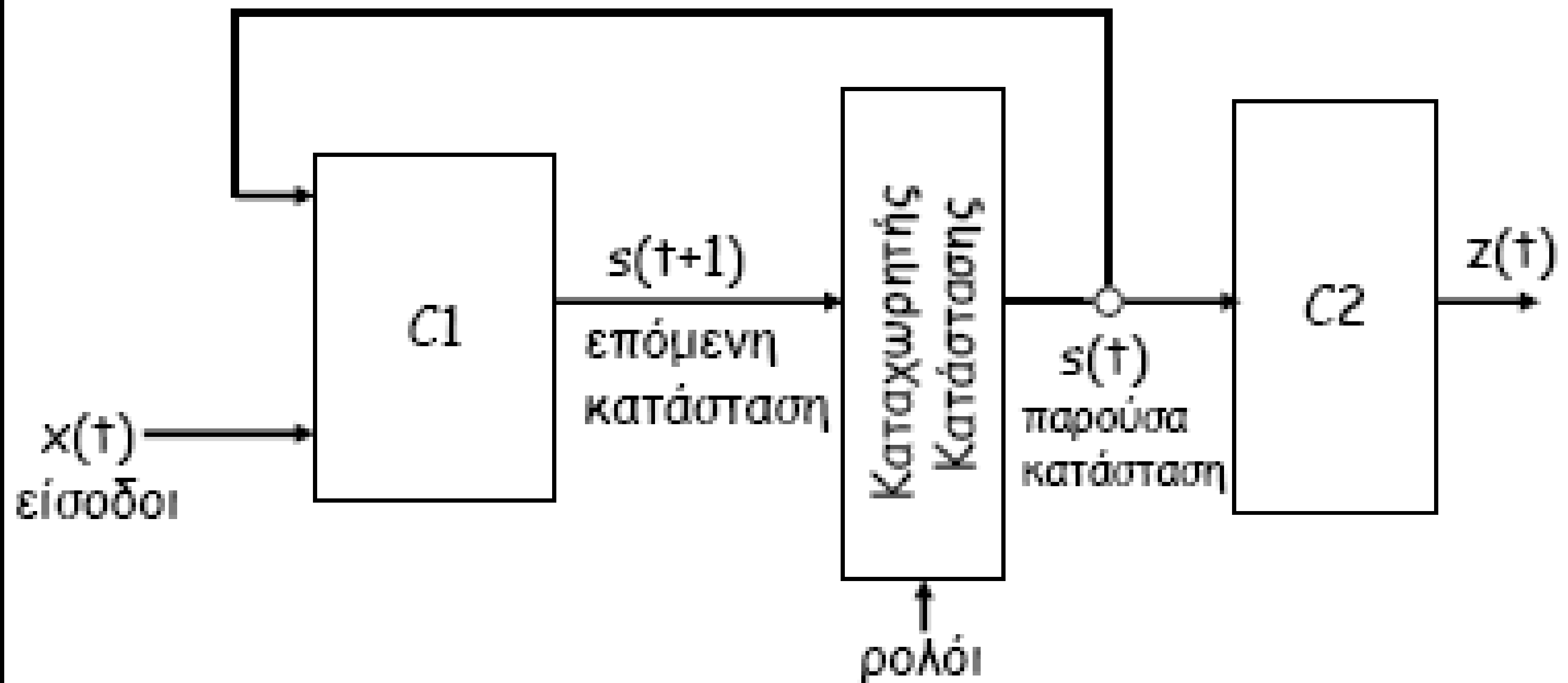
Μηχανές Mealy και Moore

- Μοντέλο Mealy:
  - Έξοδοι ΚΑΙ επόμενη κατάσταση εξαρτούνται άμεσα από τις τιμές των εισόδων ΚΑΙ της παρούσας κατάστασης.
- Μοντέλο Moore:
  - ΜΟΝΟ η επόμενη κατάσταση εξαρτάται άμεσα από τις τιμές των εισόδων ΚΑΙ της παρούσας κατάστασης. Οι τιμές στις εξόδους εξαρτούνται μόνο από την παρούσα κατάσταση (δεν εξαρτούνται άμεσα από τις τιμές των εισόδων)



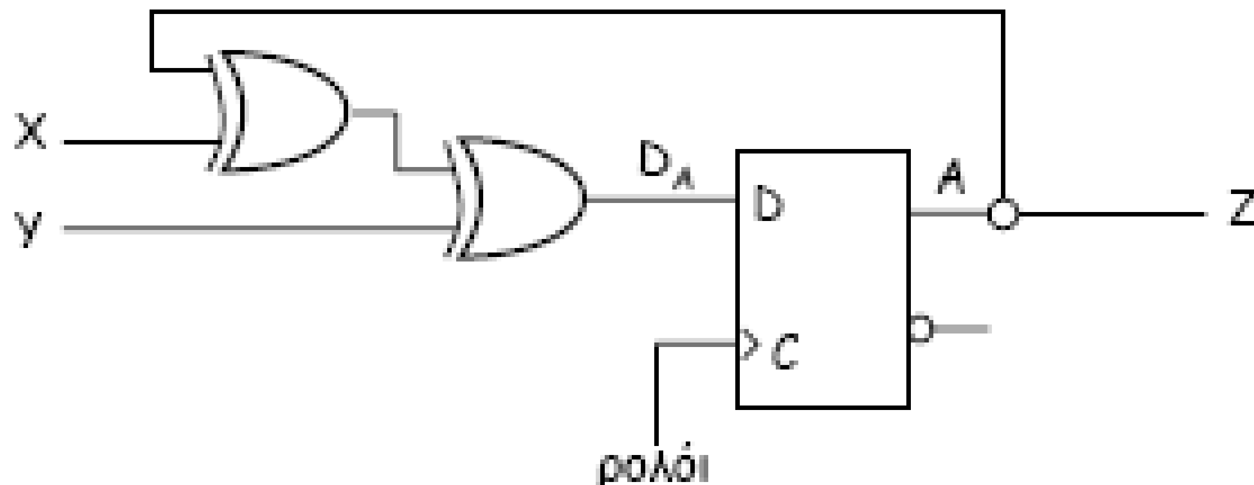
Δομή Κανονικού
Ακολουθιακού Κυκλώματος
Συνδυαστικό Κύκλωμα
s(t+1)
επόμενη κατάσταση
Καταχωρητής κατάστασης (state register - FFs)
s(t)
παρούσα κατάσταση
x(t)
είσοδοι
ρολόι
z(t)
έξοδοι


Μηχανή Mealy
s(t+1)
C1
επόμενη
κατάσταση
Καταχωρητής
Κατάστασης
s(t)
παρούσα
κατάσταση
x(t)
είσοδοι
z(t)
C2
ρολόι


Μηχανή Moore
C1
s(t+1)
επόμενη
κατάσταση
Καταχωρητής
Κατάστασης
s(t)
παρούσα
κατάσταση
C2
z(t)
x(t)
είσοδοι
ρολόι

Παράδειγμα Μηχανής Moore

- Βρείτε το λογικό διάγραμμα και τον πίνακα καταστάσεων για:
  - $D_A = A \otimes X \otimes Y$
  - $Z = A$



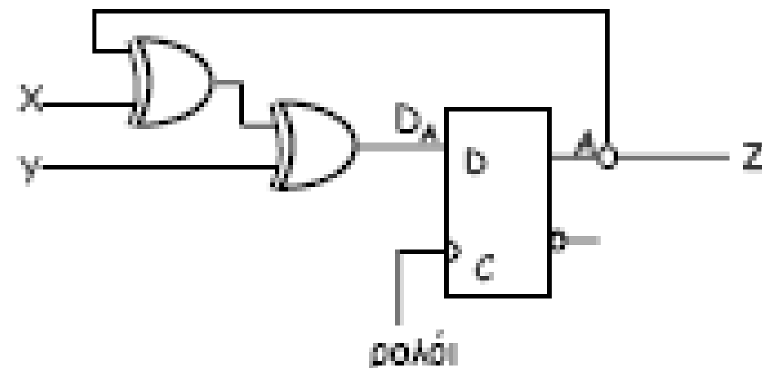
Παράδειγμα Μηχανής Moore (συν.)

Πίνακας Καταστάσεων

Παρούσα Κατάσταση	Είσοδοι		Επόμενη Κατάσταση	Έξοδος
A(t)	X	Y	A(t+1)	Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Εναλλακτική Μορφή

Παρούσα Κατάσταση	Επόμενη Κατάσταση				Έξοδος
	XY=00	XY=01	XY=10	XY=11	
A(t)	A(t+1)	A(t+1)	A(t+1)	A(t+1)	Z
0	0	1	1	0	0
1	1	0	0	1	1



- ΤΕΛΟΣ ΥΛΗΣ